# Proceedings of the KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff Conference Centre Banff, Alberta, Canada November 2-7, 1986

Sponsored by the American Association for Artificial Intelligence

Workshop Co-chairs: John H. Boose, Boeing Computer Services Brian R. Gaines, University of Calgary

## Proceedings of the KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

## Banff Conference Centre, Banff, Alberta, Canada November 2-7, 1986

## Sponsored by the American Association for Artificial Intelligence



These proceedings are provided by Boeing Computer Services in a limited distribution for workshop participants. Please do not reproduce any portion of this work without written permission from the authors.

KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP Sponsored by the American Association for Artificial Intelligence Banff, Canada, November 2-7, 1986

- ...

## CONTENTS

Program	iv
List of Participants and Authors	ix
Technical Papers (listed alphabetically by author)	
"The KREME Knowledge Editing Environment," Glenn Abrett, Mark H. Burstein, BBN Laboratories	1-0
"Ontological Analysis: An Ongoing Experiment," James H. Alexander, Michael J. Freiling (speaker), Sheryl J. Shulman, Steven Rehfuss and Steven L. Messick, Tektronix Laboratories	2-0
"Knowledge Elicitation Using Discourse Analysis," Nicholas J. Belkin, Rutgers University; H. M. Brooks, The City University, London; P. J. Daniels, Ministry of Defense, U.K.	3-0
"On the Structure of Expository Text: Preliminaries Building an Expert System from Manuals," Genevieve Berry-Rogghe, Temple University, Randy M. Kaplan	4-0
"Expertise Transfer and Complex Problems Using AQUINAS as a Knowledge Acquisition Workbench for Expert Systems " John H. Boose, Jeffrey M. Bradshaw (speaker), Boging — tificial Intelligence Center	5-0
"Induction of Horn Clauses: Methods and the Plausible Generalisation Algorithm," Wray Buntine, New South Wales Institute of Technology and Macquarie University	6-0
"Generic Tasks in Knowledge-Based Reasoning: The 'Right' Level of Abstraction for Knowledge Acquisition," Tom Bylander, B. Chandrasekaran, Ohio State University	7-0
"Acquisition of Uncertain Rules in a Probabilistic Logic," John G. Cleary, University of Calgary	8-0
"Cognitive Biases and Corrective Techniques; Proposals for Improving Elicitation Procedures for Knowledge-Based Systems," David A. Cleaves, USDA Forest Service	9-0
"The Application of Psychological Scaling Techniques to Knowledge Elicitation for Expert Systems," Nancy M. Cooke, James E. McDonald, New Mexico State University	10-0
"A Foundational Approach to Autonomous Knowledge Acquisition in Knowledge-Based Systems," James P. Delgrande, Simon Fraser University	11-0
"KRITON: A Knowledge Acquisition Tool for Expert Systems," Joachim Diederich, Ingo Ruhmann, Mark May, German Research Institute for Mathematics and Data Processing	12-0
"MOLE: A Tenacious Knowledge Acquisition Tool," Larry Eshelman, Damien Ehret, John McDermott, and Ming Tan, Carnegie-Mellon University	13-0

"An Overview of Knowledge Acquisition and Transfer," Brian Gaines, University of Calgary	14-0
"Knowledge Based Knowledge Acquisition for a Statistical Consulting System," William A. Gale, AT+T Bell Laboratories	15-0
"A Conceptual Framework for Knowledge Elicitation," Chaya Garg-Janardan and Gavriel Salvendy, Purdue University	16-0
"Design for Acquisition: Principles of Knowledge System Design to Facilitate Knowledge Acquisition," Thomas Gruber, Paul Cohen, University of Massachusetts	17-0
"Structured Analysis of Knowledge," Simon A. Hayward, STC Technology Limited, B. J. Wielinga and J.A. Breuker University of Amsterdam	18-0
"Specification of Expertise: Knowledge Acquisition for Expert Systems," Paul E. Johnson, School of Management; Sharon Gruber, 3M Company	19-0
"Knowledge Acquisition for Fault Isolation Expert Systems," Kenneth De Jong, George Mason University, and The Navy Center for Applied Research in AI	20-0
"An Intelligent Mixed-Initiative Workbench for Knowledge Acquisition," Gary S. Kahn, Edwin H. Breaux, Robert L. Joseph, and Peter DeKlerk, Carnegie Group Inc.	21-0
"Heuristics for Expertise Transfer <sup>.</sup> The Automatic Management of Complex Knowledge Acquisition Dialogs," Catherine M. Kitto, John H. Boose, Boeing Artificial Intelligence Center	22-0
"KNACK - Report Driven Knowledge Acquisition," Georg Klinker, Joel Bentolila, Serge Genetet, Michael Grimes, and John McDermott, Carnegie Mellon University	23-0
"Generalization in a Noisy Environment: The Need to Integrate Symbolic and Numeric Techniques in Learning Yves Kodratoff and Michel Manago, Laboratoire de Recherche en Informatique, Univ Paris-Sud and CNRS; Jim Blythe and Clive Smallman, GEC Research, Thionry Andry, Cognitach	94-0
"YAKYAK: Yet Another Kit for Your Acquisition of Knowledge," James Kornell, General Research Corp.	24-0
"The Knowledge Acquisition Grid: A Method for Training Knowledge Engineers," Marianne LaFrance, Boston College	26-0
"On Competence and Performance Notions in Expert System Design A Critique of Rapid Prototyping" Otto E. Laske, Arthur D. Little, Inc.	27-0
"Modeling Human Expertise in Knowledge Engineering: Some Preliminary Observations," David C. Littman, Yale University	28-0
"Taking Backtracking with a Grain of SALT," Sandra Marcus, Boeing Artificial Intelligence Center	29-0

./

• • •

"Measurement of Declarative and Procedural Knowledge in the Development of a Knowledge-Based Media Planning System " Andrew A. Mitchell, University of Toronto	30-0
"INFORM: An Architecture for Expert-Directed Knowledge Acquisition," Eric A. Moore, Schlumberger / Applicon, Alice M. Agogino, Univiversity of California, Berkeley	31-0
"Acquiring Domain Models," Katharina Morik, Technische Universitat Berlin	32-0
"Using A Domain Model to Drive an Interactive Knowledge Editing Tool," Mark A. Musen, Lawrence M. Fagan, David M Combs, Edward H. Shortliffe, Stanford University School of Medicine	33-0
"Explanation-Based Learning for Knowledge based Systems," Michael J. Pazzani, The Aerospace Corp. and the UCLA Artificial Intelligence Laboratory	34-0
"Analysis of the Performance of a Genetic Algorithm-Based System for Message Classification in Noisy Environments," Elaine J. Pettit, Merit Technology, Inc.; Michael Pettit, University of Texas at Dallas	35-0
"Simplifying Decision Trees," J. R. Quinlan, New South Wales Institute of Technology	36-0
"Multiple Problem Spaces in the Knowledge Design Process," Alain Rappaport, Neuron Data and the Robotics Institue, Carnegie-Mellon University	37-0
"Ontology and Inventory: A Foundation for a Knowledge Acquisition Methodology," Steven Regoczei, Trent University; Edwin P. O. Plantinga, University of Toronto	38-0
"Techniques for Knowledge Acquisition and Transfer," Mildred L. G. Shaw and Brian R Gaines, University of Calgary	39-0
"Knowledge Base Debugging Using Apprenticeship Learning Techniques," David C. Wilkens, Stanford University	40-0
"Case Generation for Rule Synthesis," Edward Wisniewski, Brown University; Howard Winston, Reid Smith, Michael Kleyn, Schlumberger Doll Research	41-0
"Mapping Cognitive Demands and Activities in Complex Problems Solving Worlds," David D. Woods, Westinghouse Research and Development Center, Erik Hollnagel, Computer Resources International, Copenhagen	42-0

#### PROGRAM

### KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Sponsored by the American Association for Artificial Intelligence

Banff, Canada, November 2-7, 1986

Co-chairmen:

John H. Boose, Boeing Artificial Intelligence Center Brian R. Gaines, University of Calgary Program Committee Jeffrey Bradshaw, Boeing Computer Services William Clancey, Stanford University Cathy Kitto, Boeing Computer Services Janusz Kowalik, Boeing Computer Services John McDermott, Carnegie-Mellon University Ryszard Michalski, University of Illinois, Urbana Art Nagai, Boeing Computer Services Mildred Shaw, University of Calgary

#### **DAILY SCHEDULE**

Workshop sessions will be held in the residence building. There will be coffee and juice breaks each morning and afternoon, and lunch will be served from 12:00 - 1:00. Lunches and continental breakfasts will be served in the residence building; hot breakfasts and dinners will be available in a separate Centre building.

#### SCHEDULE OF EVENTS

#### Sunday, November 2

2:00 - 9:00: Registration 7:00: Reception

. . . .

#### Monday, November 3

#### **Plenary Sessions**

Plenary speakers will describe the state of the art from their point of view, and give a short synopsis of each paper being presented in their topic area.

8:30 - 9:30:	"An Overview of Knowledge Acquisition and Transfer," Brian Gaines, University of Calgary
9:30 - 10:30:	"Cognition and Expertise," William Clancey, Stanford University
10:30 - 11:00:	Break
11:00 - 12:00:	"Interactive Interviewing Tools I," John McDermott, Carnegie-Mellon University
12:00 - 1:00:	Lunch
1:00 - 2:00:	"Interactive Interviewing Tools II," John Boose, Boeing Artificial Intelligence Center
2:00 3:00:	"Approaches to the Analysis of Knowledge Structures," Gavriel Salvendy, Purdue University
3:00 - 3:30:	Break
3:30 - 4:30:	"Learning," Ryszard Michalski, University of Illinois at Urbana
	Dinner 6.30 Raws 4/5

## Tuesday, November 4

(Note: presenters should allow at least 10 minutes for questions during their presentation time.)

## Room A

## **Cognition and Expertise**

8.30 - 9:00	"Modeling Human Expertise in Knowledge Engineering: Some Preliminary Observations," David C. Littman, Yale University
9:00 9:30	"Cognitive Biases and Corrective Techniques; Proposals for Improving Elicitation Procedures for Knowledge-Based Systems," David A Cleaves, USDA Forest Service
9:30 10:00	"Mapping Cognitive Demands and Activities in Complex Problems Solving Worlds," David D Woods, Westinghouse Research and Development Center; Erik Hollnagel, Computer Resources International, Copenhagen
10:00 - 10:30	Break
10:30 - 11:00:	"Explanation-Based Learning for Knowledge-based Systems," Michael J. Pazzani, The Aerospace Corp. and the UCLA Artificial Intelligence Laboratory
11:00 -11:30:	"Ontology and Inventory: A Foundation for a Knowledge Acquisition Methodology," Steven Regoczei, Trent University; Edwin P. O. Plantinga, University of Toronto
11:30 - 12:00:	"Acquiring Domain Models," Katharina Morik, Technische Universitat Berlin
12:00 - 1:00:	Lunch
Approaches	to the Analysis of Knowledge Structures

1:00 - 1:30:	"Knowledge Elicitation Using Discourse Analysis," Nicholas J. Belkin, Ruigers University; H. M. Brooks, The City University, London; P. J. Daniels, Ministra of Defense, U.K.
1:30 - 2:00:	"Specification of Expertent Knowledge Acquisition for Expert Systems," Paul E. Johnson, School of Management; Sharon Gruber, 3M Company
2:00 - 2:30:	"The Application of Psychological Scaling Techniques to Knowledge Elicitation for Expert Systems," Nancy M. Cooke, James E. McDonald, New Mexico State University
2:30 - 3:00:	Break
3:00 - 3:30:	"A Conceptual Framework for Knowledge Elicitation," Chaya Garg-Janardan and Gavriel Salvendy, Purdue University
3:30 - 4:00.	"Structured Analysis of Knowledge," Simon A. Hayward, STC Technology Limited; B. J. Wielinga and J. A. Breuker, University of Amsterdam
4 00 4:30	"Measurement of Declarative and Procedural Knowledge in the Development of a Knowledge-Based Media Planning System," Andrew A. Mitchell, University of Toronto
4:30 - 5:00:	"On the Structure of Expository Text Preliminaries to Building an Expert System from Manuals," Genevieve Berry-Rogghe, Temple University; Randy M. Kaplan

#### Room B

## Interactive Interviewing Tools I

	0
8:30 - 9:00:	"The KREME Knowledge Editing Environment," Glenn Abrett Mark H. Burstein, BBN Laboratories
9:00 - 9:30:	"MOLE: A Tenacious Knowledge Acquisition Tool," Larry Eshelman, Damien Ehret, John McDermott, and Ming Tan, Carnegie-Mellon University
9:30 - 10:00:	"Knowledge Based Knowledge Acquisition for a Statistical Consulting System," William A. Gale, AT + T Bell Laboratories
10:00 - 10:30:	Break
10:30 - 11:00:	"An Intelligent Mixed Initiative Workbench for Knowledge Acquisition," Gary S. Kahn, Edwin H. Breaux, Robert L. Joseph, and Peter DeKlerk, Carnegie Group Inc
11:00 -11.30:	"KNACK - Report-Driven Knowledge Acquisition," Georg Klinker, Joel Bentolila, Serge Genetet, Michael Grimes, and John McDermott, Carnegie Mellon University
11:30 - 12:00	"YAKYAK. Yet Another Kit for Your Acquisition of Knowledge," James Kornell, General Research Corp.
12:00 - 1:00:	Lunch
1:00 - 1:30:	"Taking Backtracking with a Grain of SALT," Sandra Marcus, Boeing Artificial Intelligence Center
1:30 - 2:00:	"Using A Domain Model to Drive an Interactive Knowledge Editing Tool," Mark A. Musen, Lawrence M. Fagan, David M. Combs, Edward H. Shortliffe, Stanford University School of Medic ne
2:00 - 2:30:	"Multiple Problem Spaces in the Knowledge Design Process," Alain Rappaport, Neuron Data and the Robotics Institue, Carnegie Mellon University
2:30 - 3:00:	Break
3:00 - 3:30:	"Techniques for Knowledge Acquisition and Transfer," Mildred Shaw and Brian Gainesx sz, University of Calgary
Learning I	
3:30 4:00:	"Knowledge Base Debugging Using Apprenticeship Learning Techniques," David C. Wilkens, Stanford University
<b>4:00</b> · <b>4:3</b> 0:	"Case Generation for Rule Synthesis," Edward Wisniewski, Brown University; Howard Winston, Reid Smith, Michael Kleyn, Schlumberger-Doll Research

Dinner 7.00 for 4/5

## Wednesday, November 5

## Room A

2 <sup>11</sup> .

Learning II	
8:30 - 9:00:	"Simplifying Decision Trees." J. R. Quinlan, New South Wales Institute of Technology
9:00 - 9:30:	"Induction of Horn Clauses Methods and the Plausible Generalisation Algorithm," Wray Buntine, New South Wales Institute of Technology and Macquarie University
9:30 - 10:00:	"Acquisition of Uncertain Rules in a Probabilistic Logic," John G. Cleary, Unicersity of Calgary
10:00 - 10:30:	Break

10:30 - 11:00:	"A Foundational Approach to Autonomous Knowledge Acquisition in Knowledge- Based Systems," James P. Delgrande, Simon Fraser University
11:00 -11:30:	"Generalization in a Noisy Environment: The Need to Integrate Symbolic and Numeric Techniques in Learning," Yves Kodratoff and Michel Manago, Laboratoire de Recherche en Informatique, Univ. Paris-Sud and CNRS, Jim Blythe and Clive Smallman, GEC Research; Thierry Andro, Cognitech
11:30 - 12:00:	"Analysis of the Performance of a Genetic Algorithm-Based System for Message Classification in Noisy Environments," Elaine J. Pettit, Merit Technology, Inc.; Michael Pettit, University of Texas at Dallas

### Room B

. 202

Morning

## Interactive Interviewing Tools II

8:30 - 9:00	"KRITON: A Knowledge Acquisition Tool for Expert Systems," Joachim Diederich, Ingo Ruhmann-Mark May, German Research estitute for Mathematics and Data Processing
9:00 - 9:30:	INFORM: An Architecture for Expert-Directed Knowledge Acquisition," Eric A. Moore, Schlumberger / Applicon; Alice M. Agogino, Univiversity of California, Berkeley
9:30 - 10:00:	"Design for Acquisition: Principles of Knowledge System Design to Facilitate Knowledge Acquisition," Thomas Gruber, Paul Cole — University of Massachusetts
10:00 - 10:30:	Break
10:30 - 11:00:	"Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge Acquisition Workbench for Expert Systems," John H. Boutte, Jeffrey M. Bradshaw (speaker), Boeing Artificial Intelligence Center
11:00 -11:30:	"Heuristics for Expertise Transfer: The Automatic Management of Complex Knowledge Acquisition Dialogs," Catherine M Kitto, John H Boose, Boeing Artificial Intelligence Center
11:30 - 12:00:	"Knowledge Acquisition for Fault Isolation Expert Systems," Kenneth De Jong, George Mason University, and The Navy Center for Applied Research in Al

Lunch, afternoon, evening - Local Atmosphere Enjoyment and Bear Patrol

**Panel Discussions** 

## Thursday, November 6

<b>8:00 - 10:00</b>	Panel I · "Knowledge Acquisition Methodology and Training"
8:00 - 8:30	"Generic Tasks in Knowledge-Based Reasoning: The 'Right' Level of Abstraction for Knowledge Acquisition," Tom Bylander, B. Chandrasekaran, Ohio State University
8:30 - 9:00:	"Ontological Analysis <sup>.</sup> An Ongoing Experiment," James H. Alexander, Michael J. Freiling (speaker), Sheryl J. Shulman, Steven Rehfuss and Steven L. Messick, Tektronix Laboratories
9:00 - 9:30:	"The Knowledge Acquisition Grid: A Method for Training Knowledge Engineers," Marianne LaFra-ce, Boston College
9:30 - 10:00:	Discussion
10:00 - 10:30:	Break

10:30 - 12:00:	Panel II - "Reasoning with Uncertainty: Implications for Knowledge Acquisition"
Malankan	Description of the second states

Moderator: Ryszard Michalski, with Brian Gaines and J. R. Quinlan

12:00 - 1:00: Lunch

#### Afternoon Working Groups

Attendees will participate in one of several small working groups that will attempt to define the state-of-the-art and future research directions in their topic area

Cognition and Expertise, William Clancey, et al.

Interactive Interviewing Tools I, John McDermott, et al.

Interactive Interviewing Tools II, John Boose, et al.

Approaches to the Analysis of Knowledge Structures, Gavriel Salvendy, et al.

Learning, Ryszard Michalski, et al.

Knowledge Acquisiton Methodology and Training, B. Chandrasekaran, et al.

Reasoning with Uncertainty and Knowledge Acquisiton, Brian Gaines, J. R. Quinlan, et al.

Dinner 6.30 km 4/5.

#### Friday, November 7

#### **Working Group Presentations**

A spokesperson will present results from each working group

8:00 - 8:20:	Cognition and Expertise
--------------	-------------------------

8:20 - 8:40: In	nteractive	Interviewing Tools	Ι
-----------------	------------	--------------------	---

- 8:40 9:00: Interviewing Tools II
- 9:00 9:20: Approaches to the Analysis of Knowledge Structures
- 9:20 9:40: Learning 9:40 - 10:00: Knowledge Acquisiton Methodology and Training
- 10:00 10:30: Break

10:30 - 11:00 Reasoning with Uncertainty and Knowledge Acquisition

#### Summary

- 11:00 12:00: Workshop Summary, Brian Gaines
- 12:00 1:00: Lunch

## LIST OF PARTICIPANTS AND AUTHORS

95 Inditud

÷

in rone

#### KNOWLEDGE ACQUISITON FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Sponsored by the American Association for Aritificial Intelligence Banff, Canada, November 2-7, 1986

Co-chairs: John H. Boose, Brian R. Gaines

Glenn Abrett, Mark H. Burstein
 BBN Laboratories, 10 Moulton St., Cambridge, MA, 02238, USA

Alice M. Agogino Expert Systems Laboratory, Department of Mechanical Engineering University of California, Berkeley, 5136 Etcheverry Hall, Berkeley, CA, 94720, USA

James H. Alexander, Michael J. Freiling, Sheryl J. Shulman, Steven Rehfuss and Steven L. Messick Computer Research Laboratory, Tektronix Laboratories, P.O. Box 500, Beaverton, Or., 97077, USA

Thierry Andro Cognitech, 167 rue du Chevaleret, 75013, Paris, France

- Nicholas J. Belkin The School of Communication, Information, and Library Studies Rutgers University, New Brunswick, NJ, 08854, USA
- Genevieve Berry-Rogghe Temple University, Department of Computer Science Philadelphia, Pa., 19122, USA

Jim Blythe and Clive Smallman GEC Research, West Hanningfield Rd., Great Baddow, Chelmsford, England - CM2 8NH

- John H. Boose
   Boeing Artificial Intelligence Center, Boeing Computer Services
   P.O. Box 24346, Seattle, WA 98124, USA
  - Jeffrey M. Bradshaw Boeing Artificial Intelligence Center, Boeing Computer Services P.O. Box 24346, Seattle, Wa., 98124, USA
  - H. M. Brooks Department of Information Science, The City University Northampton Square, London, EC1V OHB, U.K.

Wray Buntine New South Wales Institute of Technology and Macquarie University, Computing Science, N.S.W.I.T., P.O. Box 123, Broadway, NSW 2007, Australia

- William Clancey
   Stanford University, Stanford, CA, USA
- B. Chandrasekaran and Tom Bylander
   Laboratory for Artificial Intelligence Research,
   Department of Computer and Information Science, 2036 Neil Ave. Mall,
   The Ohio State University, Columbus, Ohio, 43210, USA

John G. Cleary Department of Computer Science, University of Calgary Calgary, Alberta, Canada T2N 1N4

- David A. Cleaves USDA Forest Service, 4955 Canyon Crest Rd., Riverside, Ca., 92507, USA
- Bruce Conrad, Morris Sharp, Brian Woodward Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4
- Nancy M. Cooke, James E. McDonald
   Computing Research Laboratory and Department of Psychology New Mexico State University, Las Curces, NM, 88003, USA

P. J. Daniels

Admiralty Research Establishment, Ministry of Defense, Teddington, U.K.

James P. Delgrande School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada, V5A 1S6

 Joachim Diederich, Ingo Ruhmann, Mark May Research Division "Expert Systems", Institute for Applied Information Technology German Research Institute for Mathematics and Data Processing, Schloß Birlinghoven Postfach 1240, D-5205 Sankt Augustin 1, West Germany

Dona M. Erb Mitre Corporation, 1120 NASA Road One, Houston, Texas, 77058, USA

- Larry Eshelman, Damien Ehret, John McDermott, and Ming Tan Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 15213, USA
- Brian R. Gaines Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4
- William A. Gale
   AT&T Bell Laboratories 2C278, 600 Mountain Ave., Murry Hill, NJ, 07974, USA
- Chaya Garg-Janardan and Gavriel Salvendy
   School of Industrial Engineering, Purdue University, West Lafayette, Ind., 47907, USA
- Sharon Gruber
   3M Company, USA
- Thomas Gruber and Paul Cohen Experimental Knowledge Systems Laboratory, Department of Computer and Information Science University of Massachusetts, Amherst, Massachusetts, 01003, USA
- Simon A. Hayward
   STC Technology Limited, Six Hills House, London Road, Stevenage, Herts, SG1 1YB, U.K.

Erik Hollnagel Computer Resources International, Copenhagen, Denmark

- Paul E. Johnson School of Management, 271 19th Ave. South, Minneapolis, Minn., 55455, USA
- Kenneth De Jong
   Computer Science Department, George Mason University, Fairfax, VA, 22030, USA, and The Navy Center for Applied Research in AI, Code 5510
   Naval Research Laboratory, Washington, D.C., 20375
- Gary S. Kahn, Edwin H. Breaux, Robert L. Joseph, and Peter DeKlerk Carnegie Group Inc., 650 Commerce Ct., Station Square, Pittsburgh, Pa., 15219

Randy M. Kaplan 812 Halvorsen Dr., West Chester, Pa., 19382, USA

Catherine M. Kitto Boeing Artificial Intelligence Center, 7L-64, Boeing Computer Services P.O. Box 24346, Seattle, Wa., 98124, USA Michael Kleyn, Howard Winston, Reid Smith Schlumberger-Doll Research, Old Quarry Rd., Ridgefield, CT, 06877, USA Georg Klinker, Joel Bentolila, Serge Genetet, Michael Grimes, and John McDermott Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 15213, USA —Yves Kodratoff Inference and Learning Group, LRI (Laboratoire de Recherche en Informatique) Univ. Paris-Sud and CNRS, Bat. 490- 91405 ORSAY, France James Kornell General Research Corp., PO Box 6770, Santa Barbara, Ca., 93111, USA R. Koubek Purdue University, School of Industrial Engineering Grissom Hall, West Lafayette, Indiana, 47907, USA **Rick Koubek** School of Industrial Engineering, Purdue University, West Lafayette, Ind., 47907, USA Marianne LaFrance Department of Psychology, Boston College, Chestnut Hill, Massachusetts, 02167, USA Otto E. Laske Arthur D. Little, Inc., Artificial Intelligence Center, Acorn Park, Cambridge, Mass., 02140, USA David C. Littman Cognition and Programming Project, Department of Computer Science Yale University, New Haven, CT, 06520, USA Michel Manago, c/o Bruce Buchanan Knowledge Systems Laboratory, Stanford University 701 Welch Rd., Bldg. C, Palo Alto, CA 94304, USA Sandra Marcus Boeing Artificial Intelligence Center, Boeing Computer Services P.O. Box 24346, Seattle, WA 98124, USA Ryszard S. Michalski Department of Computer Science, University of Illinois, Urbana, Ill., 61801, USA Andrew A. Mitchell University of Toronto, Faculty of Management Studies 246 Bloor St. West, Toronto, Canada, M5S 1V4 - Eric A. Moore. Schlumberger/Applicon, 4251 Plymouth Rd., Ann Arbor, MI, 48105, USA Katharina Morik Technische Universität Berlin, Institut für Angewandte Informatik Computergestützte Informationssysteme, Projekt KIT-Lerner, Sekr. FR 5-8 Franklinstr. 28/29, D-1000 Berlin 10, West Germany Robert Muller System Designers, Pembroke House, Pembroke Broadway, Camberley, Surrey GU15 3XD, U.K. Mark A. Musen, Lawrence M. Fagan, David M. Combs, Edward H. Shortliffe Medical Computer Science Group, Knowledge Systems Laboratory Stanford University School of Medicine, Stanford, CA, 94305-5479, USA

Arthur Nagai Boeing Artificial Intelligence Center, 7L-64, Boeing Computer Services PO Box 24346, Seattle, WA 98124, USA

Michael J. Pazzani The Aerospace Corporation, P.O. Box 92957, Los Angeles, CA, 90009, USA, and UCLA Artificial Intelligence Laboratory, 3531 Boelter Hall, Los Angeles, CA, 90024, USA

Elaine J. Pettit
 Merit Technology, Inc., 17770 Preston Rd., Dallas, Texas, 75252, USA

Micchael J. Pettit

Department of Computer Science, University of Texas at Dallas, Richardson, TX, 75083, USA

Edwin P. O. Plantinga Department of Computer Science, University of Toronto, and Redeemer College, (Hamilton, Ontario), Canada

 J. R. Quinlan Artificial Intelligence Laboratory, Massachusetts Institute of Technology 545 Technology Square, Cambridge, MA, 02139, USA, and New South Wales Institute of Technology (permanent address) School of Computing Sciences, New South Wales Institute of Technology Sydney, NSW 2007, Australia

Alain Rappaport Neuron Data, 444 High Street, Palo Alto, CA, 94301, USA, and The Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA, 15213, USA

- Steven Regoczei Computer Studies, Trent University, Petersborough, Ontario, Canada K9J 7B8
- Mildred L. G. Shaw Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4

Joan Vickers Faculty of Physical Education, University of Calgary, Calgary, Alberta, Canada T2N 1N4

B. J. Wielinga and J.A. Breuker University of Amsterdam, Nerengracht 196, 1016 BS Amsterdam, The Netherlands

- David C. Wilkens
   Department of Computer Science, Stanford University, Stanford, CA, 94305, USA
- Edward Wisniewski, Howard Winston, Reid Smith, Michael Kleyn Schlumberger-Doll Research, Old Quarry Road, Ridgefield, CT, 06877-4108, USA
- David D. Woods
   Westinghouse Research and Development Center, Pittsburgh, PA, 15235, USA

Maria Zemankova University of Tennessee, Department of Computer Science Ayres Hall, Knoxville, Tenn., 37996, USA

## The KREME Knowledge Editing Environment

Glenn Abrett and Mark H. Burstein BBN Laboratories 10 Moulton Street Cambridge, MA 02238

## Abstract

One of the major bottlenecks in large-scale expert system development is the problem of knowledge acquisition: the construction, maintenance, and testing of large knowledge bases. This paper provides an overview of the current state of development of the KREME Knowledge Representation Editing and Modeling Environment. KREME is an extensible experimental environment for developing and editing large knowledge bases in a variety of representation styles. It provides tools for effective viewing and browsing in each kind of representational base, automatic consistency checking, macro-editing facilities to reduce the burdens of large scale knowledge base revision and some experimental automatic generalization and acquisition facilities.<sup>1</sup>

#### 1. INTRODUCTION

#### 1.1. The Knowledge Acquisition Problem

The creation of the large and detailed bodies of knowledge needed to substantially improve the performance of expert systems has proven to be an extremely difficult task. We have identified several factors which make building these very large knowledge bases impractical using current technology:

Knowledge comes in many forms. Human knowledge about the world comes in many forms. Squeezing all the knowledge that an expert system needs into one, or at best two, representational formalisms (e.g rules and frames) is difficult, time consuming, and usually an inadequate solution to the task at hand.

Managing large knowledge bases is difficult. As knowledge bases grow in size and complexity they strain the capacities of software tools for knowledge editing, maintenance, and validity checking. Viewpoints at the right level of detail are hard to construct, consistency checking takes up more and more time, and global reorganizations and modifications become virtually impossible to accomplish.

**Previously encoded knowledge is not re-used.** It is customary to start building a new expert system with an empty knowledge base, even though the completed knowledge base will contain at least some general knowledge about the world. To make matters

<sup>&</sup>lt;sup>1</sup>This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by RADC under contract number F30602-85-C-0005.

worse, this general world knowledge is usually entered in a fragmentary and sketchy manner. If general knowledge about the world could be transferred across systems, the gradual accumulation of detail and precision would tremendously enhance the performance of most expert systems.

#### 1.2. Overview of the BBN Knowledge Acquisition Project

Our goal has been to develop an environment in which the problems of knowledge acquisition faced by every knowledge engineer attempting to build a large expert system are minimized. To this end, we have organized the task of developing knowledge acquisition tools into two stages. During stage one, we have been developing a knowledge representation, editing and modeling environment, dubbed KREME. We believe both knowledge engineers and subject matter experts with some knowledge of basic knowledge representation techniques will find easy to use KREME to acquire, edit, and view from multiple perspectives knowledge bases that are several times larger (i.e., 5-10,000 concepts) than those found in most current systems. During phase two of the project, we will be considering more directed kinds of automatic knowledge acquisition; techniques for generating controlled acquisition dialogues, procedures to automatically transforming previously acquired knowledge for use in new tasks, techniques for learning by analogy and learning from examples.

The current version of KREME provides, within a uniform environment, a number of special purpose editing facilities that permit knowledge to be represented and viewed in a variety of formalisms appropriate to its use, rather than forcing all knowledge to be represented in a single, unitary formalism. In addition to a general editing environment, KREME provides tools to do the kinds of validation and consistency checking so essential during the development or modification of knowledge bases. As the size of knowledge bases grows, and more people become involved in their development, this aspect of knowledge acquisition becomes increasingly important. In the hybrid or multi-formalism representational systems that are becoming prevalent [Rich 82, Brachman 83, Vilain 85], techniques must be provided for consistency checking not only within a single representational system, but between related systems.

Our approach to consistency maintenance has been to develop a knowledge integration subsystem that includes an automatic frame classifier and facilities for inter-language consistency maintenance. The frame classifier automatically maintains logical consistency among all of the frames or conceptual class definitions in a KREME frame base. In addition, it can discover implicit class relationships, since it will determine when one definition is logically subsumed by another even when the knowledge engineer has not explicitly stated that relationship. The inter-language consistency maintenance facility checks for inconsistencies in references to frames in knowledge bases specified using other representation languages (e.g., rules, procedures).

A second important area of investigation in developing the KREME editing environment has been the attempt to provide facilities for large-scale revisions of portions of a knowledge base. Our experience indicates that the development of an expert system inevitably requires systematic, large scale revisions of portions of the developed representation. This is often caused by the addition or redefinition of a task the system is to perform. These kinds of systematic changes to a knowledge base have, to date, only been possible by painstaking piecemeal revision of each affected element, one at a time. Our initial approach has been to provide a macro-editing facility, in which the required editing operations can be demonstrated by example and applied to specified sets of knowledge structures automatically. We plan to provide a library of such generic macro-editing operations for the most common and conceptually simple (though potentially difficult to describe) operations during phase two of the project.

Finally, we have begun to investigate techniques for *automatic generalization* of concepts defined in a knowledge base. We will briefly describe these experiments as well, in the section on *knowledge extension*.

#### 2. OVERVIEW OF KREME

KREME attempts to deal with the inextricably related problems of knowledge representation and knowledge acquisition in a unified manner by organizing multiple representation languages and multiple knowledge editors inside of a coherent global environment. A key design goal for KREME was to build an environment in which existing knowledge representation languages, appropriate to diverse types of knowledge, could be integrated and organized as components of a coherent global representation system. The current KREME Knowledge Editor can be thought of as an extensible set of globally coherent operations that apply across a number of related knowledge representation editors, each tailored to a specific type of knowledge. Our approach has been to integrate several existing representation languages in an open ended architecture that allows the extension of each of these languages. In addition, we have provided for the incorporation of additional representation languages to handle additional types of knowledge.

KREME currently contains knowledge editors for four distinct representation languages; one for frames, one for rules, one for procedures, and one for attached behaviors or methods, defined as functions. The rule editor, the procedure editor and the functional method editor are accessible through a global mechanism that treats these types of knowledge as forms of procedural attachment to frames. In the immediate future we plan to add a language for representing causal and other qualitative constraint systems, and several types of instantiation mechanisms, including a truth maintenance system for propositional representations.

Underlying the entire system is a strong notion of meta-level knowledge about knowledge representation and knowledge acquisition. The representation languages were implemented based on a careful decomposition of existing knowledge representation techniques, implemented as combinable objects using FLAVORS [Keene and Moon 85]. By organizing this meta-level knowledge base modularly, behavioral objects implementing such notions as inheritance and subsumption could be "mixed in" to a variety of representational subsystems making the incorporation of new representations and their editors a reasonably straightforward process. That is, each object in the meta-knowledge base encodes some aspect of a traditional representational technique, and is responsible for its own display, editing and internal forms.

#### 3. THE KREME KNOWLEDGE EDITING ENVIROMENT

The KREME Knowledge editor currently consists of three major editor modules; a frame editor, a procedure editor, and a rule editor; and a large tool-box of editing techniques that are shared among the editor modules. This section will describe the global environment and toolbox, later sections will describe the individual editors.

#### 3.1. Basic Editing Environment

Each distinct type of representation included in the system (currently concepts, roles, procedures methods and rules) has defined for it one or more editor views. A view is a configured collection of windows appearing together on the screen, each of which displays some aspect of the particular piece of knowledge being edited and/or a set of editing operations on it. When the user desires to enter or edit a specific piece of knowledge, the system opens the most appropriate view for the type of knowledge and the editing operation requested. Typically, any aspect of the chunk of knowledge being edited can be changed or viewed in more detail simply by pointing at it. This organization allows knowledge to be viewed by the user from multiple perspectives at the level of detail required.

The editor maintains a level of indirection between the knowledge being edited and the representation of that piece of knowledge in the knowledge base. This is accomplished

by a mechanism like that of text editor buffers. Changes are always made to editor definition objects which are distinct from the corresponding objects in the actual knowledge base. A stack or list of the active definition objects is always visible to the user. The top item in this list is the definition currently being viewed and edited. The user is free to modify this definition in any way without directly effecting the knowledge base. When the modified definition is to be placed into the knowledge base a defining function appropriate to the type of knowledge (e.g., classification for concepts and roles), is executed and the knowledge base is modified.

Since the editor stack is always visible, it provides one convenient method for browsing. The user may make any definition item currently in the stack the top, visible item by pointing at it. The object will be displayed in the same editor view as when it was last edited.

#### 3.2. Window Editing Tools

There are a number of window subsystems or tools that have been developed and incorporated into the KREME editor to make editing, viewing and browsing in knowledge bases easier and faster. We will now briefly describe some of the most important of them.

#### The Grapher

The KREME grapher is a powerful, generalized facility that rapidly draws lattices of nodes and links. Specific kinds of graphs appear, when appropriate, in a number of editor views. At present, dynamically updated graphs display the parts of the subsumption hierarchy surrounding (i.e., all of the objects subsuming and subsumed by) the concept or role currently being edited in the frame editor. Other concepts may be added to the displayed graph at any time simply by pointing at a node that is already present and requesting all of *its* parents or children be displayed as well. The presence of these graphs greatly facilitates browsing, since any visible node can be pointed at and made the top, current definition being edited. Alternatively, the lisp form of the object's definition can be temporarily displayed over the graph. In addition to these graphs, a graph can be "popped up" to display networks other than the one normally presented.

One extremely convenient feature of the KREME grapher is that it routinely displays graphs that are much larger than the window through which it is viewed. Simply clicking and dragging the mouse across the graph window causes the grapher to smoothly pan in the direction of mouse motion, making previously obscured portions of the graph instantly visible as though one was moving a window across a larger page. Graphs can also be displayed vertically or horizontally, in a number of different fonts and sizes, and with or without a dynamic overview that provides another convenient method for scrolling. Graphs can also be hardcopied automatically.

#### Tabular Display Windows

For editing lists of structured features of objects, we have incorporated into our toolbox a sophisticated kind tabular display window facility. Information displayed in tabular windows can be scrolled both vertically and horizontally, and edited by pointing at a component or a whole row of the table. These windows appear in a number of editor views.

#### Structure/text Editing Windows

Another style of editing commonly found in LISP environments is structure editing. Some KREME Editor views contain windows of this variety, including the macro editing view (See section 7). When editing functional methods, the structure editor can also be toggled to act as a ZMACS style editor.

#### 3.3. Files and Multiple Language Support

All definitions manipulated by the editor are read and stored in lisp-readable text files of defining forms. Since these files contain formatted lisp forms, they are userreadable, and can be edited offline using an ordinary text editor. In fact, KREME can as easily read files that were developed independently using a text editor or some other frame editor.

Files are read in using the LOAD command. A file can be loaded into a blank KREME knowledge base or can be loaded on top of an already existing knowledge base. This mechanism, which relies heavily on the the frame classifier (see section 5) to maintain consistency, enables KREME to organize information from multiple knowledge bases to create a single unified whole.

KREME currently reads and writes definitions in either its own frame language syntax or NIKL syntax, the language upon which it was largely based. This flexibility has made it possible for KREME to be used regularly to examine and update a knowledge base of approximately 1000 roles and concepts for a natural language query system that was built using NIKL. KREME can also read files of MSG (the frame language of the STEAMER [Williams et al. 81] system) defining forms, providing us access to the extensive STEAMER knowledge base of concepts and procedures. We are currently at work in building an interface to files of KEE frame definitions, as well.

We feel that this multiple language handling facility is a crucial feature of KREME and are committed to extending it, where possible, to other representation languages. A rich library of input translation programs will enable a knowledge base builder, working in KREME, to draw upon many previously existing knowledge bases to create a larger and more detailed whole. It is our opinion that this kind of flexibility will be crucial if knowledge bases developed in different languages are ever to be combined and conveniently modified to create larger ones.

#### 4. THE KREME FRAME EDITOR

Much of the work done in the current implementation of KREME has been focused on building a knowledge editor for a frame representation language. This section will describe the language that the editor works with, its basic structure and operations. Section 5.1 will describe the classifier mechanism that is used within the editor to help maintain correctness and consistency of frame knowledge bases developed with the editor.

#### 4.1. The KREME Frame Language

A number of frame languages have been developed in recent years to support AI systems [Roberts and Goldstein 77, Bobrow et al 77, Sidner 81, Moser 83, Brachman 83, KEE 84, KnowledgeCraft 85]. These languages have been well researched and extensively tested, and while we had to have some frame language on which to base our initial editor, we did not want to design and implement a new one completely from scratch. Our most important criteria for a suitable frame representation language were that it:

- 1. Allowed multiple inheritance
- 2. Was a logically worked out mature language.
- 3. Had some mechanism for internal consistency checking.
- 4. Was built on a modular object oriented base so that the language could be decomposed in such a way as to make it easily extensible.

NIKL (the definitional or frame language component of KL-TWO) [Moser 83, Schmolze and Israel 83, Vilain 85] seemed an ideal candidate. It is a fully worked out frame representation language that allows multiple inheritance, is reasonably expressive and, perhaps most importantly, was designed to work effectively with an automatic classification algorithm that could be easily adapted to provide a powerful mechanism for consistency checking and enforcement during knowledge base development. However, no object-oriented implementation of NIKL existed, and the NIKL classifier was not designed to allow *modification* and *reclassification* of previously defined concepts. A second frame language, known as MSG, had been built as part of BBN's STEAMER project and was readily available. MSG is object oriented in both of the above senses but it has no classifier and is not as mature or thoroughly specified a language as NIKL.

To develop KREME, we elected to reimplement NIKL as an object oriented language using MSG as a guide. The NIKL data structures were decomposed into a modular hierarchy of flavor definitions, and the KREME frame language was then built out of these flavors. This enabled us to incorporate a great deal of the fairly sophisticated instantiation mechanism of MSG with minimal effort. In the process, we were also able to re-implement the NIKL classifier algorithm to in a more modular fashion, both to provide the kind of reclassification capability required for a knowledge editing environment, and in anticipation of extending the classifier to deal with the richer semantics of languages like Intellicorp's KEE [KEE 84].

#### 4.1.1. Frame Language Syntax

The remainder of this section will briefly describe the basic definitional syntax of the KREME Frame language. As this syntax closely resembles the formal syntax of NIKL interested readers are referred to [Moser 83] for more detail.

Following NIKL, a KREME frame is called a concept. Collections of concepts are organized into a rooted inheritance or subsumption lattice sometimes referred to as a taxonomy of concepts. A single distinguished concept, usually called THING, serves as the root or most general concept of the lattice. A concept has a name, a textual description, a primitiveness flag, a list of concepts that it specializes or is subsumed by, a list of slots, a list of slot equivalences, and a list of concepts that it is disjoint from

The lists of slots, slot equivalences and disjoint concepts are collectively referred to as the *features* of a concept. If each concept can be thought of as defining a unique category, then features of the concept define the necessary conditions for inclusion in that category. If a concept is not marked as *primitive*, the features also constitute the complete set of sufficient conditions for inclusion in that category.<sup>2</sup> A concept inherits all features from those concepts above it in the lattice (those concepts that subsume it, and, thus, are more general) and may define additional features that serve to distinguish it from its parent or parents.

Slots (sometimes called role restrictions) consist of a role or slot name, a a value restriction, a number restriction and an (optional) default form. The value restriction specifies the class of concepts allowed as values for that slot. As in NIKL, value restrictions usually specify a particular concept. We are currently in the process of extending the value restriction language to permit more complex forms containing conjunctions, disjunctions and negations, based on the restriction language for KEE frames [KEE 84]. This effort should result in an extended classifier, as well, capable of maintaining consistency among frames in the KEE class of frame languages.

Slot Equivalences describe slots (and slots of slots) that by definition must always refer to the same entities.

<sup>&</sup>lt;sup>2</sup>Concepts marked as *primitive* (sometimes referred to as *Natural Kinds*) have no complete set of sufficient conditions. For example, an ELEPHANT must, by necessity, be a MAMMAL, but without an exhaustive list of the attributes that distinguish it from other mammals, it must be represented as a primitive concept. The class of WHITE ELEPHANTs, on the other hand, might be completely described as a ELEPHANT, with slot COLOR restricted to WHITE.

The role name specified for each KREME slot refers to an object called a *role*. Roles in KREME, as in NIKL and several other frame languages like KRYPTON [Brachman 83], and KnowledgeCraft [KnowledgeCraft 85], are actually distinct, first class objects that form their own distinct taxonomy, rooted at the most general possible role, usually called RELATION. Roles describe two place relations between concepts. A *role restriction* at a concept is thus a specification of the ways a given role can be used to relate that concept to other concepts.

#### 4.2. Frame Editing Views

The frame editor is used to create, edit and browse around in a knowledge base of KREME frames. The current KREME frame editor has 4 views, each a fixed configuration of windows appearing at once on the screen. Three windows (screen regions) are common to all of these views; the global command window, the editor stack window, and the state window. Figure 4-1 shows the main concept editing view, which contains most of the windows used for editing portions of a concept's definition. The descriptions of each window below will refer to the numbers superimposed on that figure.

- 1. The **global command window** contains commands that operate on the network as a whole. It is always visible.
- 2. The editor stack window, which is also always visible, shows the names of the things being edited and some information about their current edit state (e.g., whether they have been modified). Items in the stack window can be removed from the editor, made the currently visible edit item, or reclassified (if modified) by pointing at them.
- 3. The state window, which is visible in all views for concepts and roles, displays the name, textual description, primitive class flag, parents and information on the classification state of the item.
- 4. The concept graph window displays a dynamically updated graph of all of the abstractions and specializations of the current concept. This view provides constant visual display of the relative position of the concept being edited in the subsumption hierarchy.
- 5. The tabular feature window ordinarily displays a table of (all or just the locally defined) slots for the current concept. Columns in the table show the source of the slot (where it was inherited from), its role name, value and number restrictions, default value, and a description. This window can also be used to display the the concept's slot equivalences, disjoint concepts, or a list of all *behaviors* (rules, methods and procedures) defined for the concept (See section 6).
- 6. The **slots command window** contains various commands for editing and displaying the slots of the concept.
- 7. The Editor Interaction Window is a Lisp Listener which can be scrolled backward and forward through a history of the current session.

Three other views are currently defined for concepts, and one view is defined for roles. Two of the other concept views are somewhat different configurations of the above windows, the last is the the macro and structure editing view described in section 7. A separate view exists for editing role definitions. It contains windows showing a graph of the role taxonomy, highlighting the currently visible role, a window displaying the concepts that restrict the role and a third window displaying the role's domain, range and parents.



Figure 4-1: The Main Concept Editing View

#### 4.3. Frame Editing Operations

Space does not permit a full description of the functionality of the KREME frame editor so we will very briefly touch upon a few of its more important operations.

Making new concepts. The New Concept command in the global command menu initiates the definition of a new concept that is (1) fully specified by the user, (2) similar to some already defined concept, or (3) a specialization of one or several other defined concepts. When the initial form for the new concept has been specified the system creates a new concept definition for it and shows this new definition in the main concept view. The user is then free to add details (slots, equivalences, additional parents, etc.) to the new concept definition, classify it, or edit other concepts.

Adding and modifying slots. Whenever the window displaying slots is visible, slots can be added or modified. A new slot is added to the defined slots of the concept with the *Add Slot* command. Any portion of a slot's definition can be entered by typing or by pointing to a visible reference to the desired item. When a role or concept name that is not defined is specified, the system offers to make one with the name given.

Users may modify any locally defined slot or inherited slot. Slots shown in table windows are modified by pointing at the appropriate subform and then either typing in or pointing to a replacement form. Modifying an inherited slot causes the new definition to be locally defined.

Adding and Deleting parents. The system displays the classifier determined parents of a concept in two places. The concept graph displays them as part of the abstraction hierarchy of the concept, and the state pane indicates both the defined and direct or computed parents of the concept after the word "Specializes:". Since the classifier may have found that the concept being edited specializes some concepts more specific than those given as its defined parents, defined parents that are not direct parents are preceded by a "-", while classifier determined parents that were not defined parents are preceded by a "+".

Adding new defined parents to a concept's definition is done by clicking on the word "Specializes:" in the state window and typing a concept name or pointing to any visible concept. Parents can be deleted by clicking on their names in the list of parents displayed in the state window.

**Changing names and killing concepts and roles.** KREME allows the user to change the names of concepts and roles or to delete them completely. Name changing is accomplished simply by pointing at the concept or role's name in the state window and entering a new name. The *Kill* command splices a concept out of the taxonomy by connecting all of its children to all of its parents. There are several complicating issues relating to how concepts are deleted from taxonomies that we do not have time to discuss here.

#### 5. KNOWLEDGE INTEGRATION AND CONSISTENCY MAINTENANCE

One of the most time consuming tasks in building large knowledge bases is maintaining internal consistency. Modification, addition or deletion of knowledge in one part of a knowledge base can have wide ranging consequences to both the meaning and structure of the knowledge stored in other parts of the knowledge base. A central component of the KREME system design was that it incorporate tools for consistency maintenance both within and across representation languages. These tools are collectively referred to as the *knowledge integrator*. When new knowledge is entered or existing knowledge modified it is the task of the knowledge integrator to propagate. throughout the knowledge base, the changes that this new or modified knowledge entails, and to report any inconsistencies that have been caused by the change.

In essence, the knowledge integrator takes each new or changed chunk of knowledge (e.g., a frame, role, rule or procedure) and determines, first, how the new definition

fits into the knowledge base and, second, which other definitions depend on the current one for their meaning within the knowledge base. These dependencies are placed on an agenda and which, in turn, causes them to go through essentially the same process.

The knowledge integration subsystem for frames is basically an extension of the *classification* algorithm developed for the NIKL representation language. The NIKL classifier correctly inserts *new* frames into their proper spot in a taxonomy, by finding the most specific set of concepts whose definitions *subsumed* the definition of the new concept. The KREME classifier was designed to additionally allow existing concepts and roles to be modified and and then *reclassified*, so that the effects of redefinitions are automatically propagated throughout the entire frame network. This was accomplished by redesigning the original NIKL classifier to take advantage of the meta-level descriptions of KREME Frames and implementing the new classifier using the dependency directed agenda mechanism of the overall knowledge integrator.

#### 5.1. The Frame Classifier

The remainder of this section will give a brief description of the frame classification part of the knowledge integrator, which is the most completely developed portion of the system. For a formal description of the NIKL classifier algorithm see [Schmolze and Israel 83, Schmolze and Lipkis 83]. For a more complete description of a somewhat simpler classifier for an editing environment, see [Balzac 86].

The frame classifier works in essentially two stages, starting from a concept or role definition, as supplied by the editor or read from a file. The first stage, called completion, refers to the basic inheritance mechanism used by KREME Frames to install all inherited features of a concept or role in its internal description. The completion algorithm, when given a set of defined parents and a set of defined features for an object determines the full, logically entailed set of features of that object. The second stage is the actual classification or reclassification of a role or concept. That is, the determination of the complete, most specific set of parents of the object in its respective subsumption hierarchy.

#### 5.1.1. Completion

The completion algorithm is broken up into modular chunks that correspond to the decomposition of the frame language. There is a distinct component that deals with slot inheritance, another component that deals with disjoint class inheritance, a third that deals with slot equivalence inheritance and so on. This organization makes it quite straightforward to extend the language with new features that handle inheritance in different ways.

Figure 5-1 shows some of the complexities of slot inheritance. In 5-1A, the most specific value restriction for the slot LIMBS at 4-LIMBED-ANIMAL is inherited from one parent (ANIMAL) while the most specific number restriction, EXACTLY 4, is inherited from 4-LIMBED-THING. The completion algorithm determines that the restriction for the role LIMBS at the concept 4-LIMBED-ANIMAL must be EXACTLY 4 LIMBS.

Figure 5-1B shows one case for which the effective value restriction must logically be the conjunction of several concepts. Since ANIMAL-WITH-LEGS is both an ANIMAL, and a THING-WITH-LEGS, all of its LIMBS must be both ORGANIC-LIMBs and LEGs. If the concept ORGANIC-LEG, specializing both ORGANIC-LIMB and LEG, exists when ANIMAL-WITH-LEGS is being classified, the integrator will find it and make it the value restriction of the slot LEGS at ANIMAL-WITH-LEGS. If it does not exist, the integrator stops and asks if the user would like to define it (that is, define a concept that is both an ORGANIC-LIMB and a LEG).



Figure 5-1: Two Examples of Slot Completion

#### 5.1.2. Classification

The second stage of the frame classification algorithm finds all of the most specific subsumers of the concept being defined of redefined. This is the actual classification stage, and is essentially a special-purpose tree walking algorithm.

The basic classifier algorithm takes a completed definition (that is, a definition plus all its effective, inherited features) and determines that definition's single appropriate spot in the lattice of previously classified definitions. The result of a classification is a unique set of the most specific objects that subsume the definition and a unique set of the most general objects that are subsumed by the definition. When the classified definition is installed in the lattice all the concepts that subsume its features will be above it in the lattice and all the concepts that are subsumed by its features will be below it.

The classifier is built around a modularly constructed subsumption test that compares the completed sets of features of two objects. The object being classified is repeatedly compared to other, potentially related, objects in the lattice to see whether its completed definition subsumes or is subsumed by those other objects. For one definition to subsume the other, its full set of features must be a subset of the features of the other. As with completion, subsumption testing is partitioned by feature type (i.e slot, disjoint-class etc). One object subsumes the other when all of its individual feature-type subsumption checks return EQUIVALENT or SUBSUMES, and there is at least one vote for SUBSUMES. The advantage of this kind of modular organization is extensibility. If a new feature type is added to the language one need only define a subsumption predicate for that feature, and objects having that feature will be appropriately classified.

#### 5.2. An Example of Reclassification

The power of frame reclassification in an editing environment can be illustrated with the following relatively simple example. Suppose a knowledge base developer had defined both GASOLINE-POWERED-CAR and INTERNAL-COMBUSTION-POWERED-CAR as specializations of CAR, but had inadvertently defined INTERNAL-COMBUSTION-ENGINE as a kind of GASOLINE-ENGINE. In this situation, the classifier would deduce that INTERNAL-COMBUSTION-POWERED-CAR must be a specialization of GASOLINE-POWERED-CAR, as shown in figure 5-2A, since the former restricted the role ENGINE to a subclass of the latter's restriction of the same role.



**B.** After Reclassification

#### Figure 5-2: An Example of Reclassification

Redefining INTERNAL-COMBUSTION-ENGINE as a kind of ENGINE (rather than a GASOLINE-ENGINE), and then reclassifying, causes all of INTERNAL-COMBUSTION-ENGINE's dependents to also be reclassified, including INTERNAL-COMBUSTION-POWERED-CAR. Since GASOLINE-ENGINE no longer subsumes INTERNAL-COMBUSTION-ENGINE, the restrictions for GASOLINE-POWERED-CAR no longer subsume those of INTERNAL-COMBUSTION-POWERED-CAR, and the classifier therefore finds that GASOLINE-POWERED-CAR does not subsume INTERNAL-COMBUSTION-POWERED-CAR. This is shown in figure 5-2B.

The combination of inconsistency detection during the completion phase and the automatic propagation of classification changes that occurs during reclassification makes KREME a powerful and extremely useful tool for knowledge base development and refinement. Since the effects of reclassification are *immediately* made apparent to users via the dynamically updated graph of the subsumption lattice, they sometimes find that the definitions they have provided have some unanticipated logically entailed effects on their taxonomy. Sometimes these effects are surprising, al hough correct. Other times, they lead to changes and additions which make the knowledge base more complete and correct.

#### 6. EDITING BEHAVIORAL KNOWLEDGE

KREME embodies a set of mechanisms for representing and editing behavioral knowledge. One mechanism revolves around the notion of attaching chunks of behavior to frames. Since frames can also be associated with *flavors* (see above) individual behavioral chunks can be (and are) compiled into flavor methods.

A click of a mouse button and the tabular features window in the main concept view is turned into the toplevel behavior editor. All behaviors currently defined for the concept are shown. Each has a name and a type. There are three types of behaviors currently allowed; Rules, Procedures, and Methods. Existing behaviors can be edited or new ones defined. A modified form of the Symbolics flavor examiner can be accessed to show various useful information about method combination and derivation.

Methods are simply flavor methods. Editing a method throws up a text editor window which can be interacted with in normal (Symbolics) editing style or in structure editing style. Editing or inputing a new rule packet accesses the Rule Editor. Editing or inputing a new procedure accesses the Procedure Editor.

#### 6.1. Editing Rules

The rule language used by KREME is a language called FLEX [Shapiro 84], based in large part on the LOOPS rule language. FLEX allows rules to be defined in *rule packets*, which organize sets of rules that are meant to be run together. In the KREME environment, rule packets can be attached to concepts, just as if they were functional methods. In addition, they may be inherited by more specialized concepts. FLEX incorporates a mechanism for dealing with uncertainty, based on EMYCIN [van Melle 79]. The FLEX runtime environment also provides an elementary history and tracing mechanism, and an explanation system that produces pseudo-English explanations from rule traces. For efficiency, FLEX also provides a means for rule packets to be compiled as LISP code, and run without the rule interpreter present.

The KREME rule editor is built on top of the KREME structure editor. One defines and edits rules by specifying and filling out portions of rule *templates*. The user refines these templates either by using the mouse to copy parts of existing rules or by pointing at slots to be filled and typing in the desired values. Once a rule-set has been developed, the rule editor provides commands to run packets and debug them. It can also generate traces or rule histories paraphrased in pseudo-English. Mechanisms are also provided for deleting and reordering rules, and loading and saving them from files.

The rule editor is also tied to the KREME's knowledge integration subsystem (see section 5 above). At present, all references to slots of frames made in rules are checked for validity by the knowledge integrator. If invalid, the user is alerted, and may switch, if necessary to the editing the associated frame. If the problem was simply that he/she named a non-existent slot, a valid one may be selected from a menu. In the near future, the knowledge integrator will also check such cross-references in the opposite direction, as when a slot referred to by some rules is deleted or changed in the frame editor.

#### 6.2. Procedures in the KREME Environment

An obvious weakness of many knowledge representation languages is their inability to handle declaratively expressed knowledge about procedures as partially ordered sequences of actions, particularly if that knowledge is represented at multiple levels of abstraction. Although a number of systems have been developed that do various forms of planning, [Ernst and Newell 69, Sacerdoti 74, Sacerdoti 75, Stefik 81], most have not encoded their plans in an entirely declarative or inspectable fashion. Certainly the current generation of expert system tools does not provide mechanisms geared to the description of this kind of knowledge. Although it is clear that much of an expert's knowledge about a domain is about procedures and their application, little work has been done on devising ways to capture that information directly.

The STEAMER project [Williams et al. 81] began to address the issue of declarative representations for procedures in the course of developing a mechanism to teach valid steam plant operating procedures. The representation system developed for this task

had to be directly accessible to the students who were the system's users, and it had to serve as a source of explanations when errors were made. STEAMER was able to describe these procedures, decompose them, show how they were related to similar procedures and, in general, deal with them at the "knowledge level" [Newell 81] rather than as pieces of programs or rule sets. Although the syntax of the language was quite primitive, with no provisions for branching or iteration; the mechanisms for procedural abstraction, specialization, and path or reference reformulation that formed the heart of the language seemed to form the kernel of an extremely useful representational facility.

The KREME representation language family includes a descendant of the STEAMER procedure language, built using KREME's library of knowledge representation primitives. Each KREME procedure has a name, a description, an action that the procedure is meant to accomplish, a list of steps, and a list of ordering constraints that determine the partial ordering of the steps. Steps have an action and an object which names the conceptual class of things that step acts upon. Procedures are attached to specific frames and can be "compiled" into flavor methods.

Each step in a procedure may either be a primitive action or another procedure. If the object of a step defines a procedure for the action of that step then this procedure is said to be a sub-procedure of the enclosing procedure. For example, the ALIGN procedure attached to the concept SUCTION-LINE could have a step ALIGN <PUMP>. If the concept CENTRIFUGAL-PUMP, which is the object of this step for SUCTION-LINEs, defined a procedure for the action ALIGN, then the step ALIGN <PUMP> could be expanded into the steps of the procedure for aligning a centrifugal pump.

#### 6.2.1. Procedural abstraction and structure mapping

For knowledge acquisition purposes, it would be very useful if procedures were represented in an abstraction hierarchy like that for frames. In a strong sense, it seems difficult to define exactly what it means for one abstract procedure to subsume another. However, from an acquisition standpoint, much power can be gained by allowing abstract procedures to form templates upon which more specific procedures can be built, and eventually providing tools for automatic plan refinement like those found in NOAH [Sacerdoti 75]. For example, if you have some idea about how to grow plants in general, and you want to grow tomatoes, you will use your knowledge about growing plants in general as a starting point for learning about growing tomatoes. The final procedure for growing tomatoes will include some (presumably more detailed) versions of steps in the more general procedure, and may also include steps that are analogous to those used in growing other plants for which more detailed knowledge exists.<sup>3</sup>

The KREME Procedures editor has a mechanism for building templates of new procedures out of more abstract procedures. When a new procedure is being defined at a concept, the procedural abstraction function determines whether any of that concept's parents have a procedure for accomplishing the same action. If so, an initial procedure template is built by combining the steps and constraints of all the inherited, more abstract procedures. The paths (objects) of the steps are adjusted, using the concept's slot equivalences, to use "local" slot names, as much as possible, As yet this facility does not have the ability to do detailed reasoning with constraints on steps, as NOAH does. We expect to greatly expand this capability in the future.

The KREME Procedures language is currently being refined for use in the development of a new training system at BBN. That system will teach diagnostic procedures for the

<sup>&</sup>lt;sup>5</sup>For a detailed discussion of related issues see Carbonell [Carbonell 86] on derivational analogical planning.

maintenance of a large electronics system. We expect that KREME will greatly ease the knowledge acquisition problems faced by the developers of that system. It will also provide the first serious test of the effectiveness of the KREME acquisition environment in general.

#### 7. LARGE-SCALE REVISIONS OF KNOWLEDGE BASES

It seems fairly clear that as knowledge bases grow larger, and the sets of tasks that expert systems are called upon to perform expands, system developers will need better, more automatic methods for revising and reformulating their accumulated knowledge bases of representations. Toward this end, we feel that it is important to find ways of expressing and packaging conceptually clean *reformulations* of sets of frames and other representations, and begin developing facilities supporting the generation of new representations from old ones.

We are taking two different approaches to this problem. First, we have developed a macro facility for reformulations that can be expressed as sequences of standard, low-level editing operations. This facility allows users to define editing macros that can be applied to sets of frame definitions by giving a single example. Second, we are building a small library of functions providing operations that cannot be defined simply as sequences of low level editing operations. Our main purpose is to collect and categorize these utilities, and explore their usefulness in a working environment. Our hope is that a large fraction of these operations can be conveniently described using the macro facility, as it is more accessible to an experimental user community than any set of "prepackaged" utilities, and can be more responsive to the, as yet, largely unknown special needs of that community

#### 7.1. The Macro and Structure Editor

One of the views available when editing concepts in KREME is the macro and structure editor. This view (See figure 7-1.) provides display and editing facilities for concept definitions, based loosely on the kind of structure editor provided in many LISP environments. The view provides two windows for the display of stylized defining forms for concepts. The current edit window displays the definition of the currently edited concept (the top item on the editor stack). The display window is available for the display of any number of other concepts. Any concept which is visible in either window can be edited, and features can be copied from one concept to another by pointing. Both windows are scrollable to view additional definitions as required.

There is a menu of commands for displaying and editing definitions that includes the commands Add Structure, Change Structure, Delete Structure, Display Concept and Clear Display. Arguments (if any) to these commands may be described by pointing or typing. Thus, to delete a slot, one simply clicks on Delete Structure and the display of the slot to be deleted. Adding a structure is done by clicking on Add Structure, the keyword of the feature class of the concept one wishes to add to (e.g., Slot:). The new slot itself may be copied from a displayed concept by pointing, or a new one may be entered from the keyboard. Changing (that is, replacing) a structure can be done by pointing in succession at the Change Structure command, the item to be replaced, and the thing to replace it with. In most cases, Change Structure can also be invoked simply by pointing at the structure to be replaced, without the menu command.

The last two commands in the structure view's main menu provide the means to change what is displayed in the display window. Pointing at **Display Structure** and then at any visible concept name places the definition of that concept in the display window. **Clear Display** removes all items from the display window. Individual concepts can be deleted from the display window by pointing at them and clicking. The **Edit Concept** command is used to change what is displayed in the current edit window. Editing a new concept moves the old edit concept to the bottom of the display window.



Figure 7-1: The Macro Structure Editor View

#### 7.2. Developing Macro Editing Procedures

These operations, together with the globally available commands for defining new concepts and making specializations of old concepts essentially by copying their definitions, provide an extremely flexible environment in which to define and specify modifications of concepts with respect to other defined concepts. Virtually all knowledge editing operations can be done by a sequence of pointing steps using the current edit window and the display window. This style of editing is also used in the rule editor (See section 6.1.). The combination of editing features and mouse-based editor interaction style provides an extremely versatile environment for the description, by example, of a large class of editing macros.

In order to have macros, defined essentially by example, work on concepts other than those for which they were defined, the operations recorded cannot refer directly to the concepts or objects which were being edited when the macro was defined. This is handled by a kind of implicit variablization, where the objects named or pointed to are replaced by references to their relationship to the initially edited object. In most cases, these indirect references can be thought of as references to the *location* of the object in the structure editor's display windows. In fact, each new object that is displayed or edited in the course of defining a macro is placed on a stack called the *macro items list*, together with a pointer to the command that caused the item to be displayed. The utility of this form of reference will become clearer with an example.

#### 7.2.1. Macro Example: Adding Pipes Between Components

When the STEAMER [Williams et al. 81] system was developed, a structural model of a steam plant was created to represent each component in the steam plant as a frame, with links to all functionally related components (e.g., inputs and outputs) represented as slots pointing at those other objects. So, for example, a tank holding water to be fed into a boiler tank through some pipe that was gated by a valve was represented as

a frame with an OUTPUT slot whose value was a VALVE. The OUTPUT of that VALVE was a BOILER-TANK. The pipes through which the water was conveyed were not represented since they had no functional value in the simulation model. If it had become important to model the pipes, say because they introduced friction or were susceptible to leaks or explosions, then the representational model that STEAMER relied on would have required massive revision. Each component object in the system would have needed editing to replace the objects in its INPUT and OUTPUT slots with new frames representing pipes that were in turn connected by their OUTPUT slots to the next component in the system.

Load Saved Network New Concept New Save Network Edit Concept	w Role Paramete It Role Reset	ers. Generalizer
Classify Conception Kill Conception New Related	Concepts Change View	<ul> <li>(0, m) FIFE0</li> </ul>
Concept: PIPE0 Prinitive: YES Specializes: PIPE Description:	[Unclassified; Modified	C: [C:U] THUND: C: [C:U] THUND: C: [C:U] THUND:
		Editor Stack
Add Structures Change Structures Delete	Structure Display Cor	cept when the Clear Display
Cuttent Edit Item	Disploy	of Keloted Items
Concept PIPE0 Primitive: Yes Abstractions: (PIPE) All Pole Pestrictions: [Name NP VP Default] ((INPUT Exactly 1 (A TANK1)) (A TANK1)) (MASS Exactly 1 (A TANK1)) (A MASS)) (COLOP-OF Exactly 1 (A MASS) (A COLOP)) (OUTPUT Exactly 1 (A TANIG-WITH-INPUT) (A THING-WITH-INPUT))) Edvivalences: Disjoint Classes:	Concept IAHH 1 Primitive: No Abstractions: (IAHH) Pole Pestrictions: [Name NP ((COLOP-OF Exactly 1) <u>(COUTPUT Exactly 1)</u> Equivalences: Disjoint Classes:	VP Default] A VELLOW) (A VELLOW)) WALWEST (A WALWESTS)
Berne Define Macro-Masser Run Macro States Union Displa	ay Macros Anno Load Macr	rose deservers to be Maps Edited
More objects Insert a pipe between two connected devices 1. Make a new concept which specializes PIPE, named by generating a 2. Change the INPUT value restriction of item 1 to item 0. 3. Change the OUTPUT value restriction of item 1 to the OUTPUT value	a number suffix. ue restriction of item 0.	AC-ColeMa [course] TAN#1 [current concept] PIPE0 [operation 1]
Maçı'n 'remaer'		

While Editing TANK1:

Click on Define Macro. (Makes Macro Item 0 = TANK1).

- 1. Make a new concept which specializes PIPE. (Creates PIPEO as item 1).
- 2. Change the INPUT value restriction of item 1 (PIPEO) to item 0 (TANK1).
- 3. Change the OUTPUT value restriction of item 1 (*PIPE0*) to the OUTPUT value restriction of item 0 (*OUTPUT of TANK1 = VALVE1*).
- 4. Classify the current edit concept (Defines PIPEO).
- 5. Change the OUTPUT value restriction of item 0 (= VALVE1) to item 1 (PIPE0).
- 6. Classify item 0 (TANK1).
- 7. Edit the OUTPUT value restriction of item 1 (Creates item 2 = VALVE1).
- 8. Change the INPUT value restriction of item 2 (INPUT of VALVE1 = TANK1) to item 1 (PIPE0).
- 9. Classify all items.

#### Figure 7-2: Steps in PIPE Macro

One of our goals in developing the KREME macro editor was to be able to make such changes, which are simple to describe but require many tedious editing operations to accomplish, given the number of concepts affected. Figure 7-2 shows a macro that can be applied to all objects in a system with INPUT and OUTPUT slots, in order to generate and insert PIPEs into those slots. The macro also sets the OUTPUTs of those PIPEs to be the concept that was the old value of the OUTPUT slot in the concept edited, and similarly redoes all INPUTs.

1-16

Figure 7-2 shows how the macro is defined, by editing a representation of a tank (TANK1) connected (by role OUTPUT) to a valve (VALVE2). The sequence of steps required, defined only using the mouse, is shown in figure 7-2, as they would appear in the *Macro Definition* window of the editor.

Work on macro editing has really just begun. However, it already shows promise as a method for accomplishing a number of large scale restructurings of knowledge bases which are relatively simple to describe, but tedious to perform. As it stands, the system is already powerful enough to describe a number of transformations between semantically equivalent though functionally and syntactically distinct representations. Macros can also make use of the knowledge integrator to discover relationships in the knowledge base and exploit them. We are building a library of these macro operations so that other users of KREME will not be required to reinvent them.

We see our investigation of macro editing as only the first step in developing a knowledge reformulation facility that will have and make use of more understanding of the logical structure of the represented knowledge as well as providing a basic means of describing procedures to manipulate the syntactic structure of knowledge representations.

#### 8. KNOWLEDGE EXTENSION

One of the tasks faced by knowledge engineers in developing robust computerized knowledge bases is getting experts to express their often unconscious generalizations about their domains of expertise. While much of the detailed information about particular problems can be accessed and represented by looking at specific examples and problems, the expert's abstract classification of problem types and the abstract features he uses to recognize those problem types are less readily available. Experienced knowledge engineers are often able to discover and define useful generalizations that help organize the knowledge described by a human domain expert. The expert, although not previously aware of such a generalization, will often immediately perceive its relevance to and existence within his own reasoning processes, going so far as to suggest improvements, related generalizations, more abstract generalizations and so forth.

Our initial experiment in knowledge-base extension has been the development of a *frame generalization* algorithm which we will briefly describe. The building of a robust knowledge extension system that can deal with hybrid representation systems in intelligent and useful ways will form the bulk of the work to be done in the next generation KREME system. With our current generalizer, potentially useful generalizations are found by searching for sets of concept features that are shared by several unrelated concepts. Finding concepts with a given set of features is relatively easy since KREME indexes all concepts under each of its features.

When the generalizer finds a set of at least k features shared by at least m concepts, where k and m are user settable parameters, the system forms the most specific concept definition that would enclose all of the features but would still be more general than any concept in the set. Since our simple algorithm has no other external notion of "interestingness" it simply displays this potential new concept definition to the user. For example, given three concepts that are all ANIMALs and independently define the slot WINGS, the generalizer would suggest forming a specialization of ANIMAL with the slot WINGS, that these concepts would all specialize. If the user wanted to introduce this concept, he would respond by naming the new generalization, call it FLYING-ANIMAL, which would then be classified and integrated with the network. The features that are enclosed by this new, more general concept, are automatically removed from each of the more specific concepts being generalized.

#### 9. CONCLUSION

The goal of the BBN Labs Knowledge Acquisition Project is to build a versatile experimental computer environment for developing the large knowledge bases which future expert systems will require. We are pursuing this goal along two complementary paths. First, we have constructed a flexible, extensible, Knowledge Representation, Editing and Modeling Environment in which different kinds of representations (initially frames, rules, and procedures) can be used. We are now using this environment to investigate acquisition strategies for a variety of types and combinations of knowledge representations. In building and equipping this "sandbox", we have been adapting and experimenting with techniques which we think will make editing, browsing, and consistency checking for each style of representation easier and more efficient, so that knowledge engineers and subject matter experts can work together to build with significantly larger and more detailed knowledge bases than are presently practical.

Now that we are well along in constructing a first, experimental version of the editing environment, we are beginning to address the second aspect of our research plan, the development of more automatic tools for knowledge base reformulation and extension. An important part of this endeavor is the discovery, categorization and use of explicit knowledge about knowledge representations; methods for viewing different knowledge representations, techniques for describing knowledge base transformations and extrapolations, techniques for finding and suggesting useful generalizations in developing knowledge bases, semi-automatic procedures of eliciting knowledge from experts, and extensions of consistency checking techniques to provide a mechanism for generating candidate expansions of a knowledge base.

Our ultimate goal is to explore a number of approaches to knowledge acquisition and knowledge editing that could be incorporated into existing and future development environments, not to develop the definitive knowledge editing environment. AI is still a young field, and new knowledge representation techniques will continue to be developed for the foreseeable future. We are attempting to provide a laboratory for experimenting with new representation techniques and new tools for developing knowledge bases. If we are successful, many of the techniques developed in our laboratory will be adopted by the comprehensive knowledge acquisition and knowledge representation systems required to support the development and maintenance of AI systems in the future.

ACKNOWLEDGEMENTS: We would like to thank Richard Shapiro for his major contributions to the KREME project. Rich was largely responsible for the rule editor, the macro and structure editor and most of the user interface tools in the current version of the frame editor. We would also like to thank Dr. Ed Walker for his comments and contributions to several earlier drafts of this paper.

## References

1. Balzac, Stephen R. A System for the Interactive Classification of Knowledge. M.S. Thesis, M.I.T. Dept of E.E. and C.S., 1986.

2. Bobrow, D., Winograd, T. and KRL Research Group. Experience with KRL-0: One cycle of a knowledge representation language. Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Cambridge, MA., August, 1977.

**3.** Brachman, R.J., Fikes, R.E., and Levesque, H.J. "Krypton: A Functional Approach to Knowledge Representation". *IEEE Computer, Special Issue on Knowledge Representation* (October 1983).

4. Carbonell, Jaime G. Derivational Analogy: A theory of reconstructive problem solving and expertise acquisition. In *Machine Learning: Volume II*, Michalski, R. S., Carbonell, J. G. and Mitchell, T. M., Ed., Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986, pp. 371-392.

5. Ernst, G.W. and Newell, A. CPS: A Case Study in Generality and Problem Solving. Academic Press, New York, 1969.

6. IntelliCorp. KEE Software Development System. IntelliCorp, 1984.

7. Keene, Sonya E. and Moon, David. Flavors: Object-oriented Programming on Symbolics Computers. Symbolics, Inc.

8. Carnegie Group, Inc.. KnowledgeCraft. Carnegie Group, Inc., 1985.

**9.** Moser, Margaret. An Overview of NIKL. Section of BBN Report No. 5421, Bolt Beranek and Newman Inc., 1983.

10. Newell, A. "The knowledge level". Al Magazine 2, 2 (1981), 1-20.

11. Rich, C. Knowledge Representation Languages and Predicate Calculus: How to Have Your Cake and Eat It Too. Proc. AAAI, 1982, pp. 192-196.

12. Roberts, B. and Goldstein, I. P. The FRL Manual. A.I. Lab. Memo 409, M.I.T., 1977.

**13.** Sacerdoti, E. E. "Planning in a Hierarchy of Abstraction Spaces". Artificial Intelligence 5, 2 (1974), 115-135.

**14.** Sacerdoti, Earl D. A structure for plans and behavior. 109, SRI Artificial Intelligence Center, 1975.

15. Schmolze, J. and Israel, D. KL-ONE: Semantics and Classification. In Research in Knowlege Representation for Natural Language Understanding, Annual Report: 1 September 1982 to 31 August 1983, BBN Report No. 5421, 1983.

16. Schmolze, J.G., Lipkis, T.A. Classification in the KL-ONE Knowledge Representation System. Proc. 8th IJCAI, 1983.

17. Shapiro, Richard. FLEX: A Tool for Rule-based Programming. 5643, BBN Labs.," 1984.

18. Sidner, C.L.; Bates, M.; Bobrow, R.J.; Brachman, R.J.; Cohen, P.R.; Israel, D.J.; Webber, B.L., and Woods, W.A. Research in Knowledge Representation for Natural Language Understanding: Annual Report. BBN Report No. 4785, Bolt Beranek and Newman Inc., November, 1981.

**19.** Stefik, Mark. "Planning with Constraints: MOLGEN". Artificial Intelligence 16, 2 (1981), 111-169.

20. van Melle, W. A domain independent production-rule system for consultation programs. Proceedings of IJCAI-6, August, 1979, pp. 923-925.

21. Vilain. Marc. The Restricted Language Architecture of a Hybrid Representation System. Proceedings, IJCAI-85, International Joint Conferences on Artificial Intelligence, Inc., August, 1985, pp. 547-551.

22. Williams, M., Hollan, J., and Stevens, A. "An Overview of STEAMER: An Advanced Computer-Assisted Instruction System for Propulsion Engineering". Behavior Research Methods and Instrumentation 14 (1981), 85-90.

## **Ontological Analysis: An Ongoing Experiment**

James H. Alexander, Michael J. Freiling, Sheryl J. Shulman, Steven Rehfuss and Steven L. Messick

> Computer Research Laboratory Tektronix Laboratories PO Box 500 Beaverton, Or. 97077

#### ABSTRACT

Knowledge engineering is a complex activity which is permeated with with problems inherent in the difficulties of choosing the correct abstractions. Knowledge level analysis has been suggested as a technique to help manage this complexity. We have previously presented a methodology, called ontological analysis, which provides a technique for performing knowledge level analysis of a problem space. This paper presents the experiences we have gained with knowledge level analysis. Our experiences are reported and the criteria for a formal knowledge level analysis language are discussed.

#### 1. KNOWLEDGE LEVEL ANALYSIS

We have found that neophyte knowledge engineers often "don't know where to start." This roadblock is not usually the result of a lack of understanding about expert systems or representation schemes; but is due to confusion about the appropriate classification of the individual knowledge elements of a domain.

Currently, a knowledge engineer must rely on an intuitive, informal approach to collecting and organizing expert knowledge for a knowledge-based system. Newell (1982) proposed that there should be a *knowledge level* analysis to guide the development of AI systems, such as expert systems. Recently (Alexander *et al.*, 1986; Freiling *et al.*, 1986), we proposed a language and an informal methodology (*ontological analysis*) for use during knowledge level analysis, to allow us to study knowledge engineering. The goal of ontological analysis is to facilitate the high-level analysis of problem spaces.

We believe most development problems encountered in knowledge-based systems derive from *ad hoc* selection of abstractions. Ontological analysis provides a principled means to analyze and decompose a domain of interest. With the development of our technique, we have identified major conceptual areas which must be represented in a knowledge level analysis language and have gathered experience on the characteristics of the methodology of knowledge level analysis. In this paper we present some of the lessons we have learned. These include the advantages and disadvantages of ontological analysis, as well as some features to consider in a language for knowledge level analysis.

#### 2. AN OVERVIEW OF ONTOLOGICAL ANALYSIS.

As a preliminary step, we chose an analysis language loosely based on denotational semantics domain equations. The main goal of this exercise was to gain experience with knowledge level analysis and to determine how it differs from simply chosing a representation language. This

section details ontological analysis as it currently exists and is followed by an evaluation of the method and language.

Ontological analysis, is a technique for the preliminary analysis of a problem solving domain (Alexander *et al.*, 1986; Freiling *et al.*, 1986). An ontology is a collection of abstract and concrete objects, relationships and transformations that represent the physical and cognitive entities necessary for accomplishing a task. Complex ontologies are constructed in a three step process that concentrates first on the (static) physical objects and relationships, on the (dynamic) operations that can change the task world, and finally on the (epistemic) knowledge structures that guide the selection and use of these operations. To this end, we are developing a family of languages collectively called SPOONS (SPecification Of ONtological Structure).

The most useful and concise of the SPOONS languages is SUPE-SPOONS (SUPErstructure SPOONS; Freiling *et al.*, 1986), which is based on the domain equations of denotational semantics (Gordon 1979; Stoy 1977) and algebraic specification (Guttag and Horning, 1980). SUPE-SPOONS consists of two basic statement types:

- Domain equations: Site = Building × Campus. These statements define domains, or types of knowledge structures.\*
- Domain function declarations: add\_meeting: Meeting → [Meetings → Meetings]. These statements declare the type of specific domain elements.

The right hand side of statments can be composed of one or more domains or constant elements with operators relating these elements. Four primitive domains, STRING, BOOLEAN, INTEGER and NUMBER, are always assumed to be defined. Other primitive domains can be defined by explicit enumeration of their elements, or by open assignment to some collection of atomic elements.

The operators in the domain equations are of five types:

- Discriminated Union: D + E. Discriminated union of two domains defines that domain composed of each member of D and E, where original domain identity is preserved. (Generalization)
- Cross Product: D × E. Cross product of two domains describes a domain composed of all ordered pairs whose first element is a member of domain D and second element is a member of domain E. (Aggregation)
- Domain mapping: D → E. Mapping of one domain onto another creates a domain consisting of all functions which map domain D onto domain E.
- Collection of Sets: 2\*\*D. The domain consisting of all subsets of D.
- Collection of Ordered Sets: D\*. The domain of all ordered sequences of D.

#### 2.1 Building an Ontology

In addition to the formal notation, ontological analysis has a taxonomy of knowledge types which clarifies the function of each of the member statements. The statements fall into three groups:

Static Ontology— Defines the physical objects, or primative objects in a problem space, their properties and relationships. Ontological analysis begins with enumeration of physical objects in the problem space and identification of their inherent properties and relationships. At the level of

<sup>\*</sup>For most purposes, it suffices to think of domains as sets. A more complex semantics is needed if domains are defined recursively (Stay 1977) or with multiple equations.
the static ontology, the analysis performed is quite similar to the entity-relationship model of Chen (1976).

**Dynamic Ontology**— Defines the state space of the problem solving domain, and the actions that transform the problem from one state to another state. Problem solving is often characterized as search through a state space (Simon, 1981; Newell and Simon, 1972). Solution of a problem consists of selecting operators whose application transforms the current state into another. The dynamic ontology defines a problem space in terms of configurations of elements from the static ontology, and then defines problem operators as transformations built on the domain of problem states. The dynamic ontology defines which knowledge is unchanged throughout the problem solving process (*i.e.*, organizational charts, see the Appendix) and which knowledge changes as the problem is solved (*i.e.*, schedules and meeting plans).

**Epistemic Ontology**— Defines the constraints and methods that control the use of knowledge applied to the Static and Dynamic Ontologies. The epistemic ontology defines knowledge structures to guide the selection and use of these operations. The epistemic ontology usually contains two different types of knowledge structures. Some are used to select which operation should be performed. Others control the actual performance of certain operations. In fact, the only knowledge structures that do appear are those needed to guide the operations classified as heuristic operations in the dynamic ontology.

# 3. EVALUATION OF ONTOLOGICAL ANALYSIS

We have performed analyses on a wide range of domains including a system troubleshooting Tektronix Oscilloscopes (Alexander *et al.*, 1985; Rehfuss *et al.*, 1985; Freiling *et al.*, 1986), MYCIN's medical knowledge domain (Shortliffe, 1984), design rule checking for nMOS circuitry (Lob 1984), oscilloscope operation and an Intelligent Electronic Calendar (IEC, Staley, in press - an ontology is presented in the Appendix). With each analysis, our understanding of the domain was greatly enhanced; similarly our general understanding of the guiding principles of analysis was more focused.

# 3.1 Experiences Gained Using Ontological Analysis

The following are features of ontological analysis that have proven useful in the analysis of problem spaces.

1. Technical vocabulary is the primary source of base entities, properties and relationships key to the problem space. We begin our analysis with a paper knowledge base (Freiling et al., 1985); a description of the task domain in plain English. The paper knowledge base may come from verbal protocols (Ericsson and Simon, 1984), a textbook, a training manual or any convenient source. The technical vocabulary used in the paper knowledge base provides the initial elements of the static ontology. Physical entities are the starting point, their properties and relationships are derived later. This allows for a quick start from a concrete foundation.

2. Objects are assigned either atomic or derived status in order to clarify intended representations. Two types of object appear in an ontology: atomic and derived objects. Atomic objects (surrogates in database terminology; Codd, 1979) cannot be individuated by their properties. Instead they are individuated by identifying tokens (<a tomic >> in SUPE-SPOONS). For a meeting scheduler, for example, meetings can be atomic objects:

```
Meeting = <atomic>
Time_Of_Meeting = [ Meeting \rightarrow Time_Description ]
```

Properties of atomic objects are expressed via functions which map the objects into their property values.

*Derived objects* refer to those elements of an ontology that represent aggregations of other elements. Derived objects are individuated solely on the basis of common components. For example, consider the following derived object:

# Gregorian\_Time\_Point = Year × Month × Day × Hour × { 00, 15, 30, 45 }

Any two calendar dates are equal if they consist of the same year, the same month, hour and quartile. Only when their composite attributes are identical, are the elements themselves identical.

This distinction ameliorates the problem of representation of identity and partial specification. To be specific, derived objects must be fully specified to allow proper representation in a formal system. Conversely, atomic objects can exist, and be uniquely identified, even when nothing is known about them. Thus, objects which a system only gradually learns facts about should be represented in an atomic manner.

3. A clear distinction between intensional and extensional entities can be established. There are many cases in knowledge engineering where it is important to distinguish between representatives for the physical objects (the extension), and for descriptions or viewpoints of those objects (the intension). For the Intelligent Electronic Calendar problem (Staley, in press) it was necessary to define units of absolute time, and relate them to descriptions of time units with respect to one of many calendars (for example, to represent the following two descriptions of the same time interval: Gregorian 1986, and Japanese Showa 60).

A common way to achieve the distinction between extensional representatives of real world objects and intensional representatives of descriptions or classes of such objects is to define representatives for the extensional objects with only the bare minimum of structure. In the IEC, for instance,

> Real\_Time\_Point = INTEGER Real\_Time\_Interval = (Real\_Time\_Point × Real\_Time\_Point)

The primitive points in time are integers; point 0 is associated with 12 midnight on 1/1/1901, by the Gregorian calendar. The points are 15 minutes apart. Intervals of time can then be represented simply as a pair of points. (There is actually a bit more complexity for dealing with unbounded intervals, see Appendix).

Intensional descriptions with respect to various calendars can be constructed as necessary from different parts of the description.

Intensional\_Time\_Point = Gregorian\_Time\_Point + Japanese\_Imperial\_Reign\_Time\_Point Gregorian\_Time\_Point = Year × Month × Day × Hour × { 00, 15, 30, 45 } Japanese\_Imperial\_Reign\_Time\_Point = Era × Year × Month × Day × Hour × { 00, 15, 30, 45 }

Finally, intensional description can be related to extensional descriptions by the use of various interpretation functions.

interpret : [ Intensional\_Time\_Point → Real\_Time\_Point ]

Extensional identity of descriptions of varying sorts can then be defined as equality of the image under the relevant interpretation functions.

4. Most relevant abstractions are built through the use of generalization and aggregation. Generalization and aggregation (Smith and Smith, 1977) are common techniques for building large knowledge structures. These have a direct manifestation in our formalism as discriminated unions and Cartesian Products:

GENERALIZATION: Car = (Compact + Luxury\_Car + Truck) AGGREGATION: Car\_Assembly = (Engine × Chassis × Body × Drive \_ Train)

It is also possible to create implicit aggregations through the use of functions. Thus, the properties of an atomic object are implicitly aggregated through the fact that some particular function defines values for each.

```
Car = \langle atomic \rangle
Type = [Car \rightarrow \{Compact, Luxury_Car, Truck \}]
Has\_Engine = [Car \rightarrow Engine]
Has\_Chassis = [Car \rightarrow Chassis]
Has\_Body = [Car \rightarrow Body]
Has\_Drive\_Train = [Car \rightarrow Drive\_Train]
```

Note also that generalizations are implicit in the properties of objects as well. Instances of the domain Car, for example, can be decomposed into Compacts, etc. on the basis of common images under the Type function.

Table 1				
Glib Fragment				
<signal value=""> ::= 'HIGH' 'LOW'</signal>				
<signal> ::= 'SIGNAL-'<integer></integer></signal>				
<atomic predicate="" signal=""> ::= <signal> IS <signal value=""></signal></signal></atomic>				
<signal predicate=""> ::= <atomic predicate="" signal=""></atomic></signal>				
<pre><atomic predicate="" signal=""> 'when' <atomic predicate="" signal=""></atomic></atomic></pre>				

5. The structures built through ontological analysis highlight consistency or inconsistency of the lower level structures. (Alternatively, compositionality is an important general criteria.) An example of this concept illustrates the usefulness of our methodology. We first encountered the following problem in the process of building a semantic grammar (GLIB; Freiling *et al.*, 1984) for use in collecting knowledge about electronic instrument behavior. Table 1 shows a fragment of GLIB that can generate the following atomic signal predicate.

# SIGNAL-3 IS HIGH

Initially we assumed that this signal predicate would map signals into Boolean values. However, the semantics of two such statements combined with the connective when was not at all clear. If when was assumed to produce a Boolean itself, then the result would be returned by one of the 16 truth function of two Boolean values, clearly not what we had intended.

SIGNAL-3 IS HIGH when SIGNAL-4 IS LOW

Using domain equations to analyze the problem,

Signal = [Time  $\rightarrow$  Value] Signal\_Predicate = [ (Signal  $\times$  Signal\_Value )  $\rightarrow$  BOOLEAN ] = [ ([Time  $\rightarrow$  Value]  $\times$  Signal\_Value )  $\rightarrow$  BOOLEAN ]

we discovered that our signal predicate as defined was dropping the temporal information to and performing a global comparison with the threshold value. This problem was solved by creating a more appropriate description of the class for *signal\_Predicate*, which follows:

 $\begin{aligned} & \text{Signal_Predicate} = \\ & [(\text{Signal} \times \text{Signal_Value}) \rightarrow [\text{Time} \rightarrow \text{BOOLEAN}]] \\ & \equiv [([\text{Time} \rightarrow \text{Value}] \times \text{Signal_Value}) \rightarrow [\text{Time} \rightarrow \text{BOOLEAN}]] \\ & \text{when:} [[[\text{Time} \rightarrow \text{BOOLEAN}] \times [\text{Time} \rightarrow \text{BOOLEAN}]] \rightarrow [\text{Time} \rightarrow \text{BOOLEAN}]] \end{aligned}$ 

Thus, the comparison made by the *signal\_Predicate* is made at each instant of time, so that the result is not a single truth value computed from the whole signal, but a truth value for every time

unit of the signal. This makes it possible for when to preserve its when functional character, since the truth function (logical and) is now applied on a point by point basis. The compositional analysis of this type of problem is common to researchers familiar with the techniques of semantics and model theory (Allen, 1981). Techniques like ontological analysis make it available to knowledge engineers.

6. Use of multiple domain equations points to abstraction difficulties. Often, in the process of creating ontologies, we noticed a tendency to create multiple equations for the same domain, with each equation representing a different emphasis on the meaning of the domain. Invariably, we found that this flagged an improper description of the domain. There were three forms of of this problem:

• Equations representing successive refinement:

A: Name -> Person A: (First x Second) -> Person

These equations imply that domain Name can be further refined to (First x Second). The two forms must be reconciled so that they are identical. In this trivial example, the solution would simply be to define Name as (First x Second) and then have A use Name.

• Orthogonal definitions of a domain:

A:B->C A:D->E

In this case, A is a function which takes a B into a C, and/or a D into an E. This is not necessarily the same as:

 $A:(B+D) \rightarrow (C+E)$ 

since this would also allow two other combinations:

Orthogonal definitions may occur for several reasons:

1. B is a refinement of D and C a refinement of E. In this case the above example is another instance of refinement.

Workstation: Employee -> Desk Workstation: Executive -> Office

2. B and D are not directly comparable, but the function A is intended to apply to both. This implies a unifying abstraction, F, already exists or should be created. B and C are refinements of F. The same argument applies to the relationship of C and E, with a unifying abstraction G. An example could be:

# pay\_taxes: person -> dollars pay\_taxes: corporation -> dollars

# taxable\_entity: person + corporation (a unifying abstraction)

# pay\_taxes: Taxable\_entity -> dollars

3. B and D are not comparable at all and no unifying abstraction seems appropriate (their join is Top). In this case the semantics of the function A is unknown. It may be intended to be a polymorphic function.

However, if B and D cannot be rephrased as a common domain (unifying abstraction), and similarly for C and E, then the function A cannot be typed correctly using both domain equations. In that case, A has been over-generalized and should be rephrased as two functions, A1 and A2.

# A1: B->E A2: D->C

The appropriateness of polymorphism may be further investigated in the future.

The above cases could be viewed as constraints. Each of the equations must hold true. The ambiguity must be resolved without invalidating either one.

• Different views of the same object:

Syntactic rule: R: <if part> x <then part> Operational rule: R: matching\_condition -> executable\_part Semantic rule: R: premise -> action Rule abstraction: R: class\_of\_premise x class\_of\_action

In this case, a rule is typed differently depending on its use. The syntactic rule is the sense needed for parsing. The operational rule is the sense needed for the inference engine. The semantic rule is the sense needed for the knowledge base. The rule abstraction is the sense needed where the knowledge base itself reasons about its own rules.

This example has an implied abstraction which incorporates the different rule senses, for example:

Rule: Syntactic\_form x Operational\_form x Semantic\_form x Data\_form

The implied abstraction should be made explicit, and the appropriate form extracted where needed. This forces attention as to which rule sense subsequent domains must incorporate. It also identifies the parts of the system and allows their relationship to be made explicit.

# 3.2 Disadvantages to Ontological Analysis

The SUPE-SPOON language provides the facility to enumerate basic types, compose new types, provide typing for functions which operate on domains, etc. However, it has several deficiencies.

1. Types are not first class objects. While types can be constructed and referenced, the type itself is not a value. Because the issue of typing is so closely involved in the abstraction process, types as values is particularly important.

2. The type hierarchy is not available for inspection. This structure defines the existing abstraction environment and needs to be available for inspection, modification, etc.

3. There is no operational component identified. The operations of a type are necessary for fully identifying a type. We have no way to define the behavior of an object, only its structure. Which leads to:

4. There is no facility for identifying inheritance. Subtypes are commonly identified so that types may inherit operations, structure, or other relevant aspects of the type.

5. The abstraction process must allow for contradictory typing while the knowledge level analysis is in progress. Some of the issues mentioned on multiple domain equations arise because we use the multiple domain equations to represent the ongoing definition of the domains. As our knowledge becomes more precise, we create new (perhaps multiple) equations which must be reconciled.

# 3.3 The Next Generation Knowledge Level Analysis Language

This preliminary knowledge level analysis language has provided several pointers as to the next generation. The ultimate goal of knowledge level analysis is to identify the appropriate abstractions to the domain. While none of the above deficiencies is insurmountable, it is unlikely that they can be solved solely in a language. The solution to many of these problems must lie in a hybrid system of a formal language with a supporting computer environment.

The formal language and support environment must address the above disadvantages. In particular, our current thoughts are that an analysis system should include:

1. Formal and/or mechanized support for abstraction creation. Types appear to be a convenient formal mechanism for abstraction. Using a type system as a formal base, tools can be created to support the abstraction process. For example, the maintenance of a type hierarchy and the incorporation of incremental type definitions.

2. Support for managing change. Because the abstraction process is iterative, the system needs to provide support for changing abstractions, maintaining different versions, analyzing the existing abstraction structure (type hierarchy), etc.

3. Support for moving from the analysis stage (the type hierarchy) to a target representation language.

These areas are currently under investigation.

## 4. CONCLUSIONS

We have presented a technique, ontological analysis, that has provided us with much needed experience in a knowledge engineering methodology. There are weaknesses with the Ontological Analysis technique as currently defined as discussed above. Even so, we have found the methodology useful for conceptualizing a knowledge engineering problem, and creating a forum for cogent discussion. We hope that methodologies of this type will release the discipline from *ad hoc* descriptions of knowledge and provide a principled means for a knowledge engineer and expert to analyze the elements of a problem space and communicate the analysis to others. The abstract level at which domain equations characterize the semantics of structures and procedures, without specifying too much detail, help in this regard.

The effectiveness of a technique depends critically on the formulation of more and better principles to guide its use as well as a well founded language and support system. We are currently working on the formal language, tools, and environment. We invite other knowledge engineers to try this approach and relate their experiences.

# 5. REFERENCES

- Alexander, J.H., M.J. Freiling, S.L. Messick & S. Rehfuss. Knowledge Level Engineering: Ontological Analysis. Applied Research Technical Report CR-86-16, April 1, 1986; Tektronix, Inc., Beaverton, OR.
- Allen, J.F. An Interval-Based Representation of Temporal Knowledge. 1981 International Joint Conference on Artificial Intelligence, Vancouver, British Columbia, Canada, August, 1981.
- Chen, P.P. The Entity-Relationship Model -- Toward a Unified View of Data. ACM Transactions on Database Systems, 1, 1, March, 1976.
- Clancey, W.J. Heuristic Classification. Artificial Intelligence, 27, 289-350, 1985.
- Codd, E.F. Extending the Database Relational Model to Capture More Meaning. ACM TODS 4:4, December 1979, 397-434.
- Ericsson, K.A. & H.A. Simon. Protocol Analysis. MIT Press; Cambridge, MA, 1984
- Freiling, M.J., J.H. Alexander, S.L. Messick, S. Rehfuss & S. Shulman. Starting a Knowledge Engineering Project -- A Step-by-Step Approach. A.I. Magazine, 6, 3, Fall, 1985.
- Freiling, M.J. & J.H. Alexander. Diagrams and Grammars: Tools for the Mass Production of Expert Systems. *First Conference on Artificial Intelligence Applications*, IEEE Computer Society, Denver, Colorado, December, 1984.
- Freiling, M.J., J.H. Alexander, D. Feucht & D. Stubbs. GLIB -- A Language for Describing the Behavior of Electronic Devices. Applied Research Technical Report CR-84-12, April 6, 1984; Tektronix, Inc., Beaverton, OR.
- Freiling, M.J., S. Rehfuss, J.H. Alexander, S.L. Messick, & S. Shulman. The Ontological Structure of a Troubleshooting System for Electronic Instruments. *First International Conference* on Applications of Artificial Intelligence to Engineering Problems, Southampton University, U.K., April, 1986.
- Gordon, M.J.C. The Denotational Description of Programming Languages. Springer Verlag; New York, NY, 1979.
- Guttag, J. & J.J. Horning. Formal Specification as a Design Tool. Xerox PARC Technical Report CSL-80-1, January, 1980.
- Hayes, P.J. Naive Physics I: Ontology for Liquids. In J.R. Hobbs & R.C. Moore (Eds.), Formal Theories of the Commonsense World. Ablex Publishing; Norwood, NJ, 1985.
- Hobbs, J.R. Ontological Promiscuity. 23rd Annual Meeting of the ACL, Chicago, July, 1985.
- Lob, C. *RUBICC: A Rule-Based Expert System for VLSI Integrated Circuit Critique*. Electronic Research Laboratory Memo UCB/ERL M84/80, University of California, Berkeley, 1984.
- Newell, A. The Knowledge Level. Artificial Intelligence, 18, pp. 87-127, 1982.
- Newell, A. & H.A. Simon. *Human Problem Solving*. Prentice-Hall: Englewood Cliffs, N.J., 1972.
- Rehfuss, S., J.H. Alexander, M.J. Freiling, S.L. Messick & S.J. Shulman. A Troubleshooting Assistant for the Tektronix 2236 Oscilloscope. Applied Research Technical Report CR-85-34; Tektronix, Inc.; Beaverton, OR; September 25, 1985.

Simon, H.A. The Sciences of the Artificial. The MIT Press; Cambridge, MA; 1981.

- Quine, W.V.O. From a Logical Point of View. Harvard University Press; Cambridge, MA; 1980.
- Shortliffe, E.H. Details of the Consultation System. Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Addison-Wesley; Reading, MA, 1984.
- Smith, J.M. & D.C.P. Smith. Database Abstractions: Aggregation and Generalization, ACM Transactions on Database Systems, 2:2, June, 1977.
- Staley, J.L. An Intelligent Electronic Calendar: A Smalltalk-80® Application. *Tekniques*, in press, Information Display Group, Tektronix, Wilsonville, OR.

# Appendix: An example ontological analysis: The Intelligent Electronic Calendar

The IEC (Staley, in press) is an calendar system designed to automatically schedule meetings and rooms. A more complete description of the system and analysis can be found in Alexander *et al.* (1986).

Static Ontology

Person = <atomic> Persons = 2\*\*Person Project = <atomic> Department = <atomic> Scheduled\_Meeting = (Meeting × Person) Meeting\_Room = <atomic> Name = <string> Group = <atomic>

Meeting\_Room\_Accessory = {blackboard, screen, overhead\_projector} Chair\_Arrangement\_Type = {conference\_table, classroom, auditorium}

Meeting = <atomic> Meetings = 2 \*\* Meeting Required\_Participations = [Meeting → Person\_Description] Purposes = [Meeting → Meeting\_Purpose]

Meeting\_Purpose = One\_Time\_Meeting\_Purpose + Repetitive\_Meeting\_Purpose One\_Time\_Meeting\_Purpose = {discuss, plan, review} × Project Repetitive\_Meeting\_Purpose = { staff, reading, project } × Department

Meeting\_Proposal = Time\_Proposal + Location\_Proposal + Participant\_Proposal

Time\_Proposal = Time\_Description Location\_Proposal = Location\_Description Participant\_Proposal = Persons

Reflection = [Scheduled\_Meeting  $\rightarrow$  Meeting]

Person\_Name = [Person  $\rightarrow$ Name] = [Name  $\rightarrow$  Person] Person\_Attribute = Name + [Name × Hierarchical\_Link]

- + [ $[rep_of] \times Group$ ]
- + [{resp\_rep\_of} × Group] + [{head\_of} × Group]

Person\_Description = 2\*\*Person\_Attribute

Hierarchical\_Link = {boss\_of, subordinate\_of}\* Organization\_relation\_of\_Person = Hierarchical\_Link

Concession\_Type = {time, location, ... } ????? Owes\_Concession\_To = [(Person × Person) → Concession\_Type\*] negotiating\_points : [(Person × Person × Concession\_Type\* × Organization\_Relation\_of\_Person) → INTEGER

Group\_Contained\_By = [Group  $\rightarrow$  Group]

 $Member_Of_Group = [Person \rightarrow Group]$  $Project_Name = [Project \rightarrow <string>]$ 

Location\_Description = {Room\_Capacity} × INTEGER × (blackboard, no\_board) Room\_Has = Meeting\_Room → 2\*\*Meeting\_Room\_Accessory Room\_Capacity = Meeting\_Room → INTEGER Chair\_Arrangement\_In\_Room = Meeting\_Room → Chair\_Arrangement\_Type Building = <atomic> Campus = <atomic> Site = Building × Campus At = [Meeting\_Room → Site]

#### Static Ontology: Time Description

Real\_Time\_Point = INTEGER Time\_Quantum = INTEGER ; in quarter hours Quarter = { 00, 15, 30, 45 } Hour = {0..24} Date = {1..31} Month = {1..12} Year = {-BB .. +BB} Cycle = {-BB' .. +BB'} Year' = {000, 100, ..., 900} Ap = {1..13} Day = {1..28}

Identified\_Time\_Interval = [Real\_Time\_Point → INTEGER] ; interval 0 is 12 mid-12:15 on 1/1/1901 Calendar = [Real\_Time\_Interval → Calendar\_Interval] Real\_Time\_Interval =

[Real\_Time\_Point × Real\_Time\_Point]

+ [Real\_Time\_Point × {unbounded}]

+ [{unbounded} × Real\_Time\_Point]

+ [{unbounded} × {unbounded}]

Event\_Description = Interval\_Description × Meeting\_Description Interval\_Description =

[{between} × Calendar\_Point × Calendar\_Point]

+ [{before} × Interval\_Description]

+ [[after] × Interval\_Description]

+ [{before, after, during} × Event\_Description]

Calendar\_Region = ??????

Calendar\_Point = Gregorian\_Point + Japanese\_Point Calendar\_Interval = Calendar\_Point × Calendar\_Point Point\_Description =

Calendar\_Point + [{after} × Calendar\_Point]

+ [{before} × Calendar\_Point]

+ [{within} × Interval\_Description]

 $\begin{aligned} & Gregorian\_Point = Year \times Month \times Day \times Hour \times Quarter\\ & Japanese\_Point = Era \times Year \times Month \times Day \times Hour \times Quarter\\ & express\_as : [Calendar \rightarrow [Real\_Time\_Point \rightarrow Calendar\_Point]\\ & interpret\_as : [Calendar \rightarrow [Calendar\_Point \rightarrow Real\_Time\_Point] \end{aligned}$ 

Event = Scheduled\_Meeting + Block\_Schedule Events = 2\*\*Event Assignments = [Event → Real\_Time\_Interval] Schedule = Events × Assignments Block\_Schedule = {read, errand, fill\_out\_form} × Time\_Quantum

#### Dynamic Ontology

Meeting\_Plan = [ Meeting\_Proposal → Signoffs ] Signoffs = Persons Arbitrator = Person Reviewer = Person Participant = Person Old\_Meeting\_Plan = Meeting\_Plan New\_Meeting\_Plan = Meeting\_Plan

State = Meetings × Purposes × Required\_Participations

- $\times [ Meeting \rightarrow Arbitrator ] \times [ Meeting \rightarrow Reviewer ]$
- × [Meeting  $\rightarrow$  Meeting\_Plan ] × [Person  $\rightarrow$  Schedule ]
- $\times$  [Room  $\rightarrow$  Schedule ]

Operation = Heuristic\_Operation + Algorithmic\_Operation + Autonomous\_Operation + { schedule\_new\_meeting } Heuristic\_Operation = { select\_arbitrator , select\_reviewer , select\_meeting\_to\_act\_on } Algorithmic\_Operation = { create\_new\_meeting, reserve , assimilate }

Autonomous\_Operation = { signoff\_or\_propose, assent, arbitrate, initial\_proposal }

schedule\_meeting = (Purpose ×Required\_Participation) → [State → State ] create\_new\_meeting : (Purpose×Required\_Participation) → Meeting select\_arbitrator : Meeting×Purpose → Arbitrator select\_meeting\_to\_act\_on : State → Meeting select\_reviewer : Meeting → [State → Reviewer ]

 $\label{eq:constraint} \begin{array}{l} \mbox{initial\_proposal}: [\mbox{ Arbitrator} \rightarrow \mbox{Meeting\_Plan} ] \\ \mbox{signoif\_or\_propose}: [\mbox{ Reviewer} \rightarrow [\mbox{ Old\_Meeting\_Plan} \rightarrow \mbox{New\_Meeting\_Plan} ] \\ \end{array}$ 

arbitrate : (Old\_Meeting\_Plan ×New\_Meeting\_Plan)  $\rightarrow$  (Meeting\_Plan ×Continue) Continue = BOOLEAN

assent : Meeting\_Plan → [ (Person + Room) → Schedule ] assimilate : ((Meeting ×Meeting\_Plan) + ((Person + Room) ×Schedule) + (Meeting ×Arbitrator) +  $\begin{array}{l} (Meeting \times Reviewer)) \rightarrow \\ [ State \rightarrow State ] ) \end{array}$ 

# Epistemic Ontology

Arbitrator\_Selection\_Rules = 2 \*\* (Purpose ×Person\_Description)

Meeting\_Selection\_Rules = 2 \*\* Meeting\_Plan\_Pattern

Reviewer\_Selection\_Rules = 2 \*\* (Meeting\_Plan\_Pattern ×Person\_Description)

Meeting\_Plan\_Pattern = ((Time\_Pattern ×Signoff\_Pattern) (Location\_Pattern ×Signoff\_Pattern)) (Participant\_Pattern ×Signoff\_Pattern))

Time\_Pattern = Time\_Description + { anytime }

Location\_Pattern = 2 \*\* Location\_Description + { anywhere }

Participant\_Pattern = 2 \*\* Person\_Description + { anybody }

Signoff\_Pattern = 2 \*\* Person\_Description + { anybody } +
 { nobody\_but\_proposer }

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# KNOWLEDGE ELICITATION USING DISCOURSE ANALYSIS

N.J. Belkin

The School of Communication, Information, and Library Studies Rutgers University, New Brunswick, N.J. 08854 U.S.A.

#### H.M. Brooks

Department of Information Science, The City University Northampton Square, London ECLV OHB U.K.

#### P.J. Daniels

Admiralty Research Establishment Ministry of Defense, Teddington, U.K.

#### ABSTRACT

This paper is concerned with the use of discourse analysis and observation to elicit expert knowledge. In particular, we describe the use of these techniques to acquire knowledge about expert problem solving in an information provision environment. Our method of analysis has been to make audio-recordings of reallife information interactions between users (the clients) and human intermediaries (the experts) in document retrieval situ-These tapes have then been transcribed and analysed ations. utterance-by-utterance in the following ways: assigning utterances to one of the prespecified functional categories; identifying the specific purposes of each utterance; determining the knowledge required to perform each utterance; grouping utterances into functional and focus-based sequences. The long-term goal of the project is to develop an intelligent interface for document retrieval systems based on a distributed expert, blackboard architecture.

#### 1. INTRODUCTION

We are concerned with the problem of designing intelligent automated interfaces to mediate between people who feel they require information, and the (usually) computer-based knowledge resources which might contain information which could be of use to them. These three elements: user; intermediary; and knowledge resource, and the relations among them, consitute the general information system which is our focus of attention. The intermediary and knowledge resource elements together constitute the <u>information</u> <u>provision mechanism</u> (IPM). Such systems arise in, for instance, social security benefits offices, student advisory services and bibliographic retrieval systems. At present, almost all such information systems require a human intermediary. We assume that any automated interface will need to perform at least some of the functions that human intermediaries perform (as well as being capable of recognising situations when human intermediaries are necessary); such an interface would perforce be intelligent.

Human intermediaries are required in such systems for a number of reasons. Generally speaking, the users in such systems are unfamiliar with the contents, structure and access mechanisms of the data base, nor should they be required to be, for they are at best intermittent participants in any one such system. The intermediary's role in this respect is to use his/her knowledge to choose the appropriate data base(s), to structure the search appropriately to the data base, and to formulate and apply the relevant query. These tasks might seem somewhat mechanical and requiring relatively little creative 'intelligence'. However, the knowledge required for this performance is substantial, and the reasoning processes complex. More important, however, is that these tasks depend upon a specification of the user's information requirements that is unlikely to be forthcoming.

The typical situation in information systems, and indeed in many decision support systems, is that the users are unable to specify the information which might be useful in managing the problem which prompted them to enter the system. That is, they may recognise that their state of knowledge with respect to their problem or task is anomalous (see Belkin, 1980), but are usually unable to say what it will take to resolve the anomaly. Thus it appears that much of the human intermediary's task in the information interaction between user and intermediary is concerned with the building up of models of the user, the user's situation and the user's problem, which can be used as the basis for the subsequent tasks of database interrogation. This kind of activity is without doubt highly complex, requiring a wide range of learning capabilities and sophisticated reasoning mechanisms, within the context of a highly motivated and complicated interaction with the user. The advisory systems considered by Coombs and Alty (1984), although concerned with a slightly different problem domain, exhibit the same level of complexity, and the same type of interaction pattern.

In order to achieve our goal of an intelligent interface in the information system, we will need to discover and specify the necessary and sufficient <u>functions</u> of the intermediary, the <u>knowledge</u> necessary for performing those functions, and the <u>dialogue structure</u> in which those functions are performed. For implementation, we will need to use these results to determine <u>representational schemes</u> for knowledge, <u>retrieval strategies</u> and <u>general system architecture</u>, including <u>interactions</u> among the functions. These tasks cannot be accomplished by asking intermediaries and users what they are doing, or by requiring them to state their rules of action, because of the the nature of the situation in which they find themselves. Rather, it appears that

the most reasonable approach is a detailed, multi-faceted observation and analysis of the information system in action.

NAME OF FUNCTION	DESCRIPTION Determine position of user in problem treatment process, e.g. for- mulating problem, problem well- specified, etc.				
Problem State (PS)					
Problem Mode (PM)	Determine appropriate mechanism capability, e.g. document retrieval				
User Model (UM)	Generate description of user type, goals, knowledge, etc., e.g. graduate student, dissertation, etc.				
Problem Description (PD)	Generate description of problem type, topic, structure, subject, etc.				
Dialogue Mode (DM)	Determine appropriate dialogue type and level for situation, e.g. menu				
Retrieval Strategy (RS)	Choose and apply appropriate retrie- val strategies to knowledge resource				
Response Generator (RS)	Determine propositional structure of response to user				
Explanation (EX)	Describe mechanism operation, restrictions, etc., to user as appropriate				
Input Analyst (IA)	Convert input from user into structures usable by functional experts				
Output Generator (OG)	Convert propositional response to form appropriate to user, dialogue mode, etc.				

Figure 1. The functions of an intelligent interface for document retrieval. (After Belkin, Seeger, Wersig, 1983)

The knowledge we wish to elicit relates to the goals as specified. That is, we wish to know what intermediaries do, what knowledge they need and how they apply that knowledge. We need to discover how they model users and problems, what their interaction strategies are, and responses to the user. We believe, indeed, that contrary to usual advice on expert system construction, it will be necessary to spend a great deal of time and energy developing a realistic model and deep understanding of the situation before attempting implementation.

Our primary method of acquiring (or eliciting) the knowledge we require has been through functional discourse analysis of real information interactions, between human users and human intermediaries in operational information systems. This technique was based initially on a set of functions for information interaction derived from an abstract analysis of the information situation, and from our experience and observation of information interaction (Belkin, Seeger, and Wersig, 1983) and by a simulation (Belkin, Hennings, and Wersig, 1984), but the functions have been empirically confirmed through the discourse analysis itself.

The identification of these functions has led to a proposal for a 'distributed expert' architecture for the IPM (Belkin, Seeger, and Wersig, 1983), and the general notion of functional distribution and blackboard architecture, and some of the specific functions themselves, have been also suggested independently by Croft (1985). In the remainder of this paper, we explain why we chose discourse analysis as our method of knowledge elicitation, and describe the results this method has achieved.

# 2. <u>METHODS</u> OF <u>KNOWLEDGE</u> <u>ELICITATION</u> Buchanan (1983) defines knowledge acquisition as:

the transfer and transformation of problem solving expertise from some knowledge source to a program. Potential sources of knowledge include human experts, textbooks, databases and even one's own experience.

What appears on the surface to be a relatively straightforward task turns out in practice to be extremely difficult, time consuming and complex (Buchanan, 1982; Welbank, 1983). Within the field of Artificial Intelligence there is little in the way of methods or techniques that could be used to facilitate knowledge elicitation. Attention has been turned, therefore, to other disciplines where knowledge transfer is of interest, e.g. cognitive psychology.

The main techniques for knowledge elicitation can be summarized as follows:

- (a) Interviewing the expert: either informally or making use of structured interviewing techniques.
- (b) Verbal protocol analysis: that is, analysing recordings of experts thinking aloud as they carry out a task.

(c) Observational studies: observing and recording the behaviour of the expert as s/he works on a real problem, in their normal working environment, in as unobtrusive way as possible.

A review of Knowledge Acquisition methods is presented by Welbank (1983). Gammack and Young (1985) propose a number of "psychological" techniques including Personal Construct Theory (Kelly, 1955), and concept sorting. The use of Personal Construct Theory to acquire knowledge for expert system construction has also been investigated by Shaw (e.g., 1984).

Interviewing the expert is a commonly used technique. It has a number of disadvantages though, and may not be useful in areas where experts have particular difficulty in articulating their knowledge (Welbank, 1983). The use of verbal protocol analysis is also very common (e.g., Kuipers and Kassirer, 1983). Frequently the task given to the expert is a standard case study or an artificial problem. A comprehensive survey of the use of verbal protocol analysis has been carried out be Ericsson and Simon (1980). Verbal protocol analysis has also been used to investigate various aspects of the functional behaviour of librarians (Ingwersen, 1982). Observational techniques are probably the least used because they tend to be extremely time consuming and require complex, indepth analysis (Welbank, 1983). However, they do have the advantage that they can be used to discover what the expert actually does and are useful for extracting information about the role of the expert, ordering of tasks, etc. (Kidd, 1985b). One example of the use of observational techniques in the expert system context is the investigation into student-advisor interactions carried out by Coombs and Alty, with a view to developing an expert system advisor (Alty and Coombs, 1980; Coombs and Alty, 1984).

Current thinking tends towards the use of multiple techniques, with certain methods seen as more appropriate for particular types of knowledge (Gammack and Young, 1985; Kidd, 1985b). In other contexts, this has been done for some time e.g. the detailed study of librarian-user interaction carried out by Ingwersen and Kaae made use of both verbal protocol analysis and interviews (1980). Perhaps the most important factor is that whatever technique is used, it should elicit knowledge at the level of granularity required for that particular expert system application. That is, the more detailed and complex the knowledge to be elicited, the more detailed and complex the analysis must be to extract that knowledge (Clancey, 1983; Kidd and Cooper, 1985; Kidd, 1985a).

#### 3. DISCOURSE ANALYSIS FOR KNOWLEDGE ELICITATION 3.1 Introduction

The long-term aim of this project is to produce an intelligent information system interface which will simulate the functional performance of a good human intermediary. There are a number of possible methods which could be used to achieve our objective. Our approach has been to employ the methods which seemed best suited to developing a particular specification, given the problem-solving environment under investigation. These methods include:

- functional analysis of real interactions between human (a) experts (search intermediaries) and clients (information system users);
- (b) interviews with experts;
- system simulation, using humans to play the roles of the (C) various system functions;
- (đ) classification of problem and knowledge types.

The nature of the problem domain and the expert-client interaction has meant that most of our efforts have concentrated on the first of these methods, i.e. functional analysis of expert-client interactions. This method produces an extremely detailed analysis of what is, in any case, a highly complex situation, in a way that is as unconstrained as possible by pre-conceived ideas.

3.2 <u>General methodology</u> The general method in most of our research has been to collect data consisting of natural language human-human interactions. The collection of data took place at two academic online information retrieval services at London University with four different trained intermediaries, who carry out searches of biliographic databases. The users of these services are primarily postgraduate students, but also include academic staff, and other types of researchers, for example, from industry and research centres. Having obtained the permission of both participants, audiorecordings were made of six presearch interviews, from the point that the user entered the service until the beginning of the online search, when the intermediary logged onto the system.

The interviews were then transcribed from the tapes according to a specific format. The transcription of each dialogue was then checked by another of our team of three people, the transcript was divided into utterances, which are the units for analysis, and this division was again checked by another person (Daniels, Brooks and Belkin, 1985).

An utterance can be defined as a speech sequence by one participant during the conversation. It may or may not comprise of complete grammatical entities, and may be terminated by a contribution made by the other participant. If the contribution of one participant takes the conversational turn, the previous speech sequence is regarded as a completed utterance (Brooks and Belkin, 1983; Price, 1983). The six interview transcripts together comprise some 3,000 individual utterances.

Two interviews were subjected to separate analyses by three different people, each of whom attempted to identify and categorise the tasks or functions performed by utterance by both parties during the dialogue (see sections 3.3 and 3.4). The results of this suggested that analysis according to focus, and focus shift, was also possible.

The focus of a dialogue (Grosz, 1978) can be said to highlight that part of the mutual knowledge of the participants relevant at any given point in a dialogue, by grouping together those concepts or themes that are in the focus of attention. The current focus is likely to dictate the structure of the discourse, and the topics to which reference can be made at that given moment. In the analysis of our six interviews, shifts of focus wre generally indicated by the occurrence of "frame" words (Sinclair and Coulthard, 1975), often accompanied by pauses of varying duration.

Examples of frame words include:

Right OK (.) right (..) (Interview 190684HBA)

now the next thing is to get (,) our strategy (Interview 190684HBA)

Each interview was analysed in terms of foci and the goals, subgoals and functions pertaining to these foci.

#### 3.3 Adequacy of the functions

By function of an utterance, we mean to indicate WHY the participant said it; that is, what s/he intended to gain or communicate by it, what model-building activity it was relevant to, and so on. A single utterance may serve more than one function. In several successive studies we tested the adequacy of pre-determined functional patterns by attempting to categorise all of the utterances in a number of dialogues using a number of coding schemes. Each iteration used the previous results to add, delete or modify the definition of specific functions until the classification stabilized on the functions indicated in fig.1. Having analysed the interactions in this way, it was possible to obtain time-line displays of their functional structures over the course of the dialogue.

3.4 <u>Function specification</u> The individual functions necessary for successful information interaction need to be specified at a level of detail which will allow programming of each function. It is also necessary to identify and specify the knowledge resources associated with the functions, since they will require some knowledge in order to achieve their goals.

I emm (.) what's the (.) the problem /1 Ū ok I'm just beginning (.) Ι mm hm /3a research project /2 (.) err (.) I'm a research student U Ι yeah /5 U at LSE /4a (...) and um (...) working in the Geography Ι Department /4b and I'm err (.) doing a (.) thesis (.) U Ι U beginning a thesis on err (.) forestry (.) and err the interesting /6 Ι impact of recreation (.) of conflicts of recreation and U oh that's interesting (.) mm (.5 sec) I U forestry /4c Ι (inaud....) /7 TT and err (...) one of the things I'm aiming I U to do (.) eventually is to look at the cost benefits of Ι U different err (.) management schemes (.) for recreation Ι mm U (...) /8 I = intermediary /1 = utterance number 1 LEGEND: (.) = 1/2 sec pause U = userFigure 2. Transcript of focus 1, interview 120684HBA.

Each of the interviews was subjected to a detailed utterance-byutterance analysis to identify and categorise the subfunctions and subgoals that seem to be occurring. Two interviews (120684HBA and No.4) were subjected to separate analysis by three people, independently. A sample transcript with its accompanying subgoal analysis is presented above. Fig.2 is focus 1 from interview 120684HBA.

The subgoal analysis for each utterance is given below:

- 1: This is an open question from the intermediary directed towards problem description, in some form, from the user.
- 2: The user's response to the intermdiary's opening question is to describe the context of the problem, and the user's position within the problem. This utterance seems concerned with establishing the state of the user's problem.
- 4a: In this part utterance, the user describes his <u>status</u>, and the college he is attending.
- 4b: This is concerned with further user description including <u>user status</u>.
- 4c: This a composite, containing information about the user's <u>goals</u>, a re-iteration of the <u>state</u> of the problem, and finally an initial description of the user's <u>research</u>.
- 8: The last utterance in this focus involves further description by the user of his research topic.

Each of the remaining four interviews was analysed by at least two people independently. The results were cumulated for all six interactions to produce an inventory of subgoals.

Most of the subgoals appear to contribute to a particular function and could be grouped accordingly. The pattern of interaction between the subgoals associated with the User Model and Problem Description functions were analysed. Information contributed by, or donated to, other external functions was noted. These interactions were diagrammed on a focus-by-focus basis. (Examples of these interaction maps are given in Brooks, Belkin, and Daniels, 1985.)

Having identified the subgoals associated with the functions, we attempted to identify the knowledge resources which the intermediary needs to access in order to accomplish the functions during the information interaction. Utterances concerned with User Model and Problem Description subgoals were analysed in depth. Inferences were made about the knowledge which appeared to be used. For example:

User Model Knowledge:

In the following extract, the intermediary is using her knowledge of ways in which users can be classified by status and which of these categories are important in the current exchange. In this example, the intermediary knows that users can be categorised according to whether they belong to London University or are visitors and therefore, that information about this aspect needs to be elicited.

(Taken from interview 190684HBA)

I: now I gather you're .... a visitor/3 yes are you part of the university or /5

Problem Description Knowledge:

(Taken from interview No.4)

I: yeah now does it mean when people talk of community education do they mean ... primary, secondary, vocational, technical, and universities, or d- do they really mean only a certain level or type /22

The intermediary's question about the level of schooling implies some internal knowledge by the intermediary about the metastructure of the subject domain, (i.e. knowing that the topics of research projects in this area tend to be limited to a particular educational level). This knowledge of the domain meta-structure provides a framework within which to question the user about her research.

In order to verify the subgoals and knowledge resources identified for the User Model function, interviews were arranged with two experienced intermediaries at the search services from which the original data was collected. A questionnaire was devised which first asked the intermediaries to comment on the appropriateness and accuracy of the five User Model subgoals, and secondly asked for further information, confirmation, etc. of various problem areas. Each question was accompanied by a concrete example in the form of an excerpt from a transcript or a played-back recording. The information gained in these interviews confirmed the User Model analysis and clarified the problem areas.

## 3.5 Dialogue structure

Previous work (Belkin and Windel, 1984) suggested that a general problem structure for the information system domain could provide a means for driving and guiding a human-computer dialogue in this domain. We have assumed that any appropriate human-computer dialogue should follow the functional structure of the analagous human-human dialogue.

Discourse analysis of the human-human interactions was used to identify and define a problem structure associated with the information system. This identification and specification of the problem structure depended crucially on the analysis of the discourse into foci, on the specification of the goal of each focus and on an investigation of the factors initiating a shift in focus (Daniels, Brooks, and Belkin, 1985).

# 4. <u>RESULTS</u> 4.1 <u>Introduction</u>

Whilst the main emphasis of this paper is on the methodology we are using to develop the design and specification of an intelligent, distributed expert-system based interface, we would like to indicate to what effect these methods have been employed. Although the results were obtained primarily within a document retrieval environment, they also seem to apply in other sorts of advisory interaction we have studied.

# 4.2 Adequacy of the functions

Functional analysis of the six interviews has elicited a common set of functions which appear to be the minimum required for an intelligent interface in the document retrieval environment, and perhaps for information systems in general. There appears to be no obvious single order in which these functions are, or should be, performed, but rather that any sequence is circumstance driven.

Comparison of the functional structure of individual interviews has enabled us to indicate the extent to which model-building activities are carried out by both parties. In particular, the significance of the Problem Description, Problem State and User Modelling functions to the success or failure of an interaction can be demonstrated (Brooks and Belkin, 1983; Price, 1983).

#### 4.3 Function specification

For each function, it is necessary to specify the subgoals concerned with the accomplishment of that function, the knowledge resources required by that function in order to achieve its goal and the information contributed to and acquired from the other functions. Most of our efforts have concentrated on the specification of the User Model (Daniels, in press), Problem Description and Retrieval Strategies functions (Brooks, in press).

The majority of the 24 subgoals we have been able to identify in the interviews are concerned with the accomplishment of some particular function. For example, the subgoals associated with User Modelling comprise:

- UGOAL Determining the user's goals
- USER Determining the status of the user
- KNOW Determining the user's state of knowledge in the field
- BACK Determining the user's background

A complete listing of the subgoals is given in Daniels, Brooks, and Belkin, 1985.

The types of knowledge needed by the User Modelling function tend be categorical in nature. The specific knowledge used embodies a finite, restricted range of possibilites and tends to be qualitative rather than quantitative. In contrast, the knowledge needed by the Problem Description function appears much more varied and to be closely connected with the subject domain of the search topic. The knowledge elicited by the analysis for these two functions is described in more detail in Brooks, Belkin, and Daniels (1985); Daniels (in press); and Brooks (in press).

## 4.4 Dialogue structure

Given the nature of reference interviews, it must assumed that the problem structure driving the expert-client interaction in this situation is similarly complex. A uni-dimensional model would be too simplistic, and so general as to have little or no predictive value. Instead, we propose a single structure but one which is multi-dimensional and capable of representing individual aspects of an interaction whilst retaining a commonality of form (see Daniels, Brooks, and Belkin, 1985). The foci can be seen as concentrating the participants' attention on the accomplishment of a particular goal, with shifts of focus moving attention from goal to goal across the problem structure.

#### 5. DISCUSSION

As Welbank (1983) says, interviews alone cannot produce a detailed context of <u>real</u> behaviour; rather, they need to be backed up by observational studies. Our methodology concentrates on observational techniques supported by interviews. We define our methods as observational since they involve recording the expert at work on real problems in her normal environment. Verbal protocol analysis, on the other hand, requires the expert to verbalise her thoughts and actions as she carries out what is often an artificial task. Although this method has been used to study librarian-user interactions (Ingwersen and Kaae, 1980), it would have been inappropriate in our situation. We considered that interviews with intermediaries would not be productive as the principal method of knowledge elicitation either. Experts typically are unable to articulate their problem-solving expertise and this is particularly true for information provision environments where the tasks and goals of intermediaries are not wellformulated, well-defined or static. Unlike medical diagnosis for example, there is no metalanguage for describing problem-solving procedures.

We view the information interaction between user and intermediary as consisting to a large extent of model building by both participants within a cooperative dialogue. Therefore, the knowledge to be elicited must concern the way in which the participants use discourse to carry out their tasks. It is necessary then to analyse the recorded observations at the discourse level. This is the level of granularity required. The analysis that we have carried out goes beyond an analysis of broad concepts and beyond a simple functional analysis. Discourse analysis provides a "microlevel" of representation which will allow complete, complex and accurate interface implementation.

Using the methods outlined, we can also take account of the cooperative nature of the interaction, e.g. user inputs and contributions. Moreover, we have been able to outline a dialogue structure which could drive the computer half of such a dialogue. This will provide the basis for the development of a true mixedinitiative discourse component for the interface. Kidd (1985) has noted that one of the main problems with current expert systems is their rigidly constrained system-oriented dialogues.

We have also been able to propose a representation for the user model and the problem description. The user model representation consists of frames embodying information about the user's status, goals, level of knowledge, previous information system experience and general background (Daniels, in press). It is suggested (Brooks, in press) that the problem description is represented as several layers of interconnected partitioned semantic networks (Hendrix, 1979).

From the methodological point of view, we have taken great care with intercoder reliability and cross-checking. These are common procedures in psychological and sociological research but we have found very little evidence of their use in the knowledge engineering literature. Finally, it should be noted that this is an extremely timeconsuming method of analysis. The whole process, from recording, through transcription, to the analyses, involved a team of researchers over many months. On the other hand, it has provided us with sufficient detail to begin to implement an interface which will exhibit complex behaviour and be capable of sustaining mixed initiative discourse.

#### 6. CONCLUSION

The outcome of this project has not merely been to identify and specify the functions required for an intelligent interface for document retrieval systems. In addition, we feel that contributions have been made to the following:

- Knowledge elicitation methodology: Detailed discourse analysis of recorded interactions between expert and client has not been, up to now, well explored as a method for eliciting expert knowledge.

- The application of expert system techniques to complex domains: Many expert systems have been confined to very restricted, well defined domains. The application we are investigating, however, involves multi-dimensional problem-solving in a complex domain. This has implications for methodology and system architecture.

- Cognitive modelling: We regard the knowledge elicitation task as serving not only to develop an operational system but also to further our understanding of what human intermediaries do and how they do it. Interaction between human intermediaries and users involves complex cognitive modelling. Thus our research has offered insight into the development and use of multiple models by more than one party.

In conclusion, we believe that knowledge elicitation is of prime importance to expert system development and should not be regarded as secondary to the implementation of an operational system. Unfortunately eliciting knowledge at a detailed and comprehensive level is time-consuming and mitigates against the immediate implementation of expert systems.

#### 7. REFERENCES

Alty, J.L. and Coombs, M.J. (1980). Face-to-face guidance of university computer users - I: a study of advisory services. Internation Journal of Man-Machine Studies, 12, pp.390-406.

Belkin, N.J. (1980). Anomalous states of knowledge as a basis for information retrieval. <u>Canadian Journal of Information Science</u>, 5, pp.133-143.

Belkin, N.J., Hennings, R-D., and Wersig, G. (1984). Simulation of a distributed expert-based information provision mechanism. <u>Information</u> <u>Technology:</u> <u>Research</u>, <u>Development</u>, <u>Applications</u>, 3, pp.122-141.

Belkin, N.J., Seeger, T., and Wersig, G. (1983). Distributed expert problem treatment as a model of information system analysis and design. Journal of Information Science, 5, pp.153-167.

Belkin, N.J. and Windel, G. (1984). Using MONSTRAT for the analysis of information interaction. In: Dietshmann, H.J. (Ed.), pp.359-382. <u>Representation and Exchange of Knowledge as a Basis</u> of Information Processing. Amsterdam: Elsevier Science.

Brooks, H.M. (in press). Developing and using problem descriptions. In: IRFIS 6: Intelligent information systems for the information society. Frascati, Italy, Sept., 1985. North Holland.

Brooks, H.M. and Belkin, N.J. (1983). Using discourse analysis for the design of information retrieval mechanisms. In: <u>Research</u> and <u>Development in Information Retrieval.</u> <u>Proceedings</u> of the Sixth Annual International ACM SIGIR Conference, Washington, D.C., 1983, pp.31-47. New York: ACM.

Brooks, H.M., Belkin, N.J., and Daniels P.J. (1985). Problem descriptions and user models: developing an intelligent interface for document retrieval systems. In: <u>Informatics 8:</u> <u>Advances in</u> <u>Intelligent Retrieval</u>, pp.191-214. London: ASLIB.

Buchanan, B.G. (1982). New research in expert systems. In: Hayes, J.E., Michie, D., and Pao, Y.H. (Eds.), pp. 269-299. <u>Machine Intelligence 10.</u> Edinburgh: Edinburgh University Press.

Buchanan, B.G., et al. (1983). Constructing an expert system. In: Hayes-Roth, F. and Waterman, D.A. (Eds.), pp.127-168. Building Expert Systems. Reading, Mass.: Addison-Wesley.

Clancey, W.J. (1983). The epistemology of rule-based expert systems: a framework for explanation. <u>Artificial Intelligence</u>, 20, pp.215-251.

Coombs, M. and Alty, J. (1984). Expert systems: an alternative paradigm. <u>International Journal of Man-Machine Studies</u>, 20, pp.21-43.

Coombs, M. and Alty, J. (1980). Face-to-face guidance of university computer users - II: characterizing advisory interactions. <u>International Journal of Man-Machine Studies</u>, 12, pp.407-429.

Croft, W.B. (1985). An expert assistant for a document retrieval system. In: <u>RIAO 85. Actes</u> of the conference: Recherche d'Informations Assistee par Ordinateur, Grenoble, France, 18-20 March, 1985, pp.131-149. Grenoble: I.M.A.G.

Daniels, P.J. (in press). The user modelling function of an intelligent interface for document retrieval systems. In: <u>IRFIS</u> <u>6: Intelligent information systems for the information society</u>. Frascati, Italy, Sept. 1985. North Holland.

Daniels, P.J., Brooks, H.M., and Belkin, N.J. (1985). Using problem structures for driving human-computer dialogues. In: <u>RIAO 85:</u> Actes of the conference: Recherche d'Informations Assistee par Ordinateur, Grenoble, France, 18-20 March, 1985, pp.131-149. Grenoble: I.M.A.G.

Ericsson, K.A. and Simon, H.A. (1980). Verbal reports as data. Psychological Review, 87(3), pp.215-251.

Gammack, J.G. and Young, R.M. (1985). Psychological techniques for eliciting expert knowledge. In: Bramer, M.A. (Ed.), pp.105-112. <u>Research and Development in Expert Systems</u>. <u>Proceedings</u> of the 4th Technical Conference of the B.C.S. Specialist Group on Expert Systems, University of Warwick, 18-20 December, 1984. Cambridge: Cambridge University Press.

Grosz, B.J. (1978). Discourse knowledge. In: Walker, D.E. (Ed.), pp.229-346. <u>Understanding Spoken Language</u>. New York: Elsevier-North Holland.

Hendrix, G. (1979). Encoding knowledge in partitioned networks. In: Findler, N.V. (Ed.), pp.305-326. <u>Associative Networks:</u> <u>Representation and Use of Knowledge in Computers</u>. New York: Academic Press.

Ingwersen, P. (1982). Search procedures in the library analysed from the cognitive point of view. Journal of Documentation, 38(3), pp.165-191.

Ingwersen, P. and Kaae, S. (1980). <u>User-Librarian Negotiations</u> and <u>Information Search Procedures in Public Libraries: Analysis</u> of <u>Verbal Protocols</u>. Final Research Report, DB-TEK-50. Copenhagen: Royal Copenhagen School of Librarianship. Kelly, G.A. (1955). The <u>Psychology of Personal Constructs</u>. New York: Norton.

Kidd, A.L. (1985a). The consultative role of an expert system. In: Johnson, P. and Cook, S. (Eds.), pp.248-254. <u>People and</u> <u>Computers: Designing the Interface. Proceedings of the B.C.S.</u> HCI Specialist Group Conference, University of East Anglia, 17-20 Sept. 1985. Cambridge: Cambridge University Press.

Kidd, A.L. (1985b). Knowledge elicitation. B.C.S. Specialist Group on Expert Systems Lecture, London, October, 1985.

Kidd, A.L. and Cooper, M.P. (1985). Man-machine interface issues in the construction and use of an expert system. <u>International</u> <u>Journal of Man-Machine Studies</u>, 22, pp.105-112.

Kuipers, B. and Kassirer, J.P. (1983). How to discover a knowledge representation for causal reasoning by studying an expert physician. In <u>Proceedings of the 8th International Joint</u> <u>Conference on Artificial Intelligence</u>, Karlsruhe, 18-20 August, 1983, pp.49-56.

Price, L.E.T. (1983). <u>Functional and Satisfaction Analyses of</u> <u>Information Interaction Dialogues</u>. M.Sc. Thesis, Department of Information Science, The City University, London.

Shaw, M.L.G. (1984). Knowledge engineering for expert systems. In: <u>Interact</u> <u>'84</u>. <u>Proceedings</u> of the IFIP Task Group on Human-Computer Interaction, Imperial College, London, 4-7 Sept., 1984, pp.328-332.

Sinclair, J.McH. and Coulthard, R.M. (1975). <u>Towards an Analysis</u> of <u>Discourse</u>. <u>The English</u> <u>Used</u> by <u>Teachers</u> <u>and</u> <u>Pupils</u>. Oxford: Oxford University Press.

Welbank, M. (1983). A review of knowledge acquisition techniques for expert systems. <u>British Telecom Research Laboratories Report</u>, December 1983. Martlesham Heath: Martlesham Consultancy Services.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

On The Structure Of Expository Texts; Preliminaries to Building An Expert System From Manuals

Genevieve Berry-Rogghe Randy M. Kaplan

## Abstract

One task facing any knowledge engineer constructing a knowledge base and expert system is the extraction of knowledge from the domain expert and other sources. If any of this process could be automated the process of knowledge base and expert system construction would be improved. In this paper we present a framework for extracting the base or elementary knowledge present in a knowledge base from textual material. In particular, we describe how textual structure may be used for constructing predicates representing the meaning of the text being processed. Our initial source of material is an encyclopedia for children. This was chosen for several reasons. First, the sentences contained in this text do not require complex syntactic processing. Second, very little a priori knowledge on the part of the reader is assumed. These characteristics have provided a sufficiently constrained environment so as to yield some interesting advances towards automatically creating a knowledge base.

#### Keywords

Knowledge Engineering Expert Systems Natural Language Understanding Knowledge Base

#### Authors Address

Genevieve Berry-Rogghe Temple University Department of Computer Science Philadelphia, PA 19122 (215) 787-7000

Randy M. Kaplan 812 Halvorsen Drive West Chester, PA 19382 (215) 436-4570

#### On The Structure Of Expository Texts; Preliminaries to Building An Expert System From Manuals

#### Genevieve Berry-Rogghe Randy M. Kaplan

Temple University Department of Computer and Information Science

#### 1. INTRODUCTION

. 5

One of the major difficulties in constructing an expert system is the process of extracting knowledge from a human expert and encoding it in a form which is suitable for use in an expert system. A desirable goal for the future is the automatic construction of a knowledge base from technical manuals - at least for the encoding of basic general knowledge. The major purpose of this investigation is to give insight into the various processes involved in this task and to collate the efforts in text analysis and models for knowledge representation and memory processing that have been proposed in linguistics, psychology and artificial intelligence. The second part of this paper outlines an approach to text analysis for simple scientific expository texts.

In artificial intelligence, the area of natural language proessing is vastly growing and some success has been achieved in the construction of natural language interfaces to database retrieval systems. It should be pointed out, however, that the task involved here is primarily one of "transformation", namely from a sentence in English, to a formal database query, which only minimally involves real understanding of the textual material. In addition, these interfaces primarily operate on the single sentence level ignoring the complexities of connected discourse. (We should mention here some attempts to extract a knowledge base from texts, for instance [3,8] these systems are in essence also "transformation systems" working in a highly constrained textual environment.)

A system which will construct a knowledge base for an expert system from technical prose must recognize the overall structure of the text; distinguish between facts and rules and detect the relationships between sentences and between paragraphs. More relevant to this project is the research in story understanding as in Rumelhart, Schank and Charniak et al [4,16,19]. The research in this area has been primarily with the aim of explaining the inferencing mechanism and is based on matching of the text against generic knowledge schemata. Expository text, however, aims at creating knowledge schemata and hence relies to a much lesser extent on prior world A more detailed account of the characknowledge. teristics of expository text as compared to narrative text will be given in a later section.

Recently, linguists and psychologists have begun to analyze the structure of expository and tutorial texts; the aim of the psychological research has been primarily to explain the process of reading understanding in humans, whereas the linguists have studied the linguistic properties of such texts per se.

In analyzing discourse, two major levels of analysis must be distinguished: the descriptive level and the processing level. The descriptive level deals with the analysis of the structure of expository texts: how the text is segmented into elementary units and how these units combine into higher level structures.

The processing level deals with the process of text understanding: how the information is organized and stored in memory, combining it to already existing knowledge structures and how the information may be retrieved for a specific task. The next section gives an overview of various models that have been proposed for the descriptive analysis of expository texts.

#### 2. THEORIES OF DISCOURSE STRUCTURE

Most researchers in discourse analysis would agree that the descriptive analysis of a text comprises at least three levels:

- the symbolic representation level,
- the microstructure level, and
- the macrostructure level.

The symbolic representation level deals with the choice of the formal representation language for individual sentences. An obvious choice would be the predi-cate calculus. This choice has advantages, particularly the fact that it provides a well-defined semantic interpretation and offers the possibility of applying formal deductive mechanisms. The major disadvantage is that for the time being there has been no generally satisfying solution to the problem of how to represent arbitrary natural language sentences in this formalism. Notoricus are difficulties with natural language quantifiers and mass terms, both constructs occurring frequently in scientific texts. In addition, programs that attempt to model understanding established the necessity for the representation of deeper semantic relations at the lexical and clausal level. This led to the introduciton of case grammer - which is the most widely adopted model by psychologists - and the theory of "conceptual dependency" (see Schank, Abelson [18]) which became the paradigm for a significant school in artificial intelligence. Another solution which seems attractive - and which will be described in greater detail in Section 5 is to encode the text as elementary "propositions". Propositions consist of a predicate followed by a number of arguments; there is no quantification and there are no logical connectives. Propositions are connected to each other through the use of propositional variables.

This method was chosen in the work of Kintsch and van Dijk [10] who are probably the most prominent researchers in the psychological text analysis area.

The microstructural level deals with the semantic relations between various clauses in a sentence or between two succeeding sentences. For example, the two sentences "Clouds form, because water vapor in the air condenses" and "Water vapor in the air condenses so clouds form" express the same kind of causal relation between two propositions. How the relation between two propositions is represented depends to some extent on the adopted theory for the macrostructural level. Three major approaches may be distinguished: the <u>linguistic</u>-coherence approach, the <u>logical-rhetorical</u> approach and the content-based approach. The former approach seeks to elucidate the coherence of a text uniquely through the analysis of its external linguistic properties such as the presence of certain clue words, representation to lexical items and how referential expressions are used. A major propagator of this approach is the linguistic work by Halliday and Hasan [9].

The logical-rhetorical approach examines the relation between two sentences in terms of logical and rhetorical concepts such a "causation" (marked by the connectives "because, since, as"...), "contrast" (marked by "but"), "collection" (marked by "and"), "sequence" (marked by "after, before, when, whenever"...), "condition" (marked by "if...then") and so on. This approach is the most common and exemplified, among others, in the psychological research by Meyer and in the artificial intelligence research by Weiner [19] and Fox [7].

1

The content-based approach is exemplified Kintsch-van Dijk [10] and Kintsch-Miller [13]. by No explicit distinction is made here between the micro and macro structural levels. Text analysis consists of the construction of a "proposition graph". The information and relation present in the surface form the sentences is encoded as propositions (see above). The higherorder conceptual ideas derived from the text are represented by a coherence graph, which is constructed as follows. First a number of related propositions (normally corresponding to a complex sentence or two sentences at the miccrostructural level) are grouped together. Then from this set one proposition is selected on the basis of its conceptual centrality and stands at the top of the graph structure. The other propositions in the set are linked to this superordinate proposition and to each other through their shared arguments. The aim of this model was primarily to account for the limitations in short term memory processing in recall tasks. Since the major coherence establishing factor is lexical repetition, the model is less well suited to explain larger connections on the macrostruc-tural level. The method may be termed "content-based" because it explains primarily how the ideas and concepts in the text are related.

The macrostructural level deals with relationships among ideas represented in complexes of sentences or paragraphs; it explains what the major theme of the text is and how this theme is developed. Two kinds of theories about the mechanisms to determine the macrostructure of a text have been proposed. These methods may be broadly classified as either "external" or "internal".

External theories rely uniquely on the information in the text as a source. In this category fall the linguistic theories by Halliday and Hasan and the propositional graph theory explained above. Internal theories posit the existence of cognitive generic knowledge structures or schemas. Schemas are natural packets of highly structured generic knowledge which may be about world knowledge, pragmatic knowledge and knowledge about language. (Schemas have been variously referred to as "frames" - see Minsky [14] and Charniak [3], "scripts" - see Schank and Abelson [18] -"stereotypes" et cetera.)

At the macrostructural level of analysis, two types of schemata are relevant: world knowledge and text structure schemata. The frame and script based theories cited above fall into the former category whereas proposals for text grammars (see, for instance Rumelhart [18]) fall into the latter category. World knowledge schemata represent knowledge about human actions and their effects as well as specific knowledge about prototypical events in the world. This model has been quite successful in the processing of narratives and was also applied to a scientific text (about the launching of a rocket) by de Beaugrande [6]. Even though specific world knowledge may play a role in the understanding of tutorial texts - especially in specialized texts designed for experts, it would seem to be of less importance for the processing of texts imparting simple knowledge about the world - which is the kind of text it seems most appropriate to tackle initially. In the framework of our research project, we are examining in more

detail proposals about textual schemata representing generic information about the structure of expository texts, with the intention of constructing a "grammar" for macro-level analysis of such texts.

A text grammer is a schema that represents frequent configuraitons of textual elements. The basic units of the grammar may be rhetorical categories. For example, Meyer [11] proposed a text be analyzed at the macropropositional level in terms of the following five basic rhetorical categories which, recursively applied, subsume the entire macrostructure of a text:

-	collection:	related ideas or events on the the basis of some commonality
-	causation:	a causal relation between an antecedent and a con- sequent
-	response :	a problem statement followed by a solution (such as question - answer)
•	Comparison:	differences or similarities between two concepts
-	description:	giving more information about a topic (maybe by giving an example)

In Section 6.1 we shall give an example of how a text grammar may be constructed out of basic rhetorical predicates. Next, we want to examine some of the evidence given for the psychological reality of the existence of text schemata. Even though it might be argued that a system to construct a knowledge base for an expert system need not be psychologically motivated, the choice between an "external" and "internal" approach needs to be justified with respect to explanatory adequacy and processing ease.

Psychological experiments have validated the existence of text structure schemata on at least two accounts. On the one hand, it was shown that the role of content knowledge is limited when subjects are asked . to pick out the central proposition from a text. Even with very little understanding of the subject matter, subjects are usually able to recall the main statements of a text, relying on surface level linguistic and typogrampical clues which confirm their expectations generated by the internalized structure schemata. (It would appear that it is part of the self-training of college students, to cultivate the ability to process complex verbal material without fully understanding it...) In another experiment reported in Meyer [12], the same propositional content was encoded in two different ways: in the one case using the surface clues that would signal a narrative and in the other case using clues that would signal an expository text. The two texts, although having the same content, were reportedly processed differently by the subjects.

One final note on the descriptive analyses proposed. None of the above levels of analysis have been fully automated. Even at the symbolic representation level, there exists no algorithm to convert an arbitrary English sentence into a canonical representation. The main reason being that some of the major stumbleblocks in natural language processing, notably, the resolution of referential expressions has not been solved in a general way. At the microstructural level, it might seem that a system such as Kintsch's would be particularly amenable to algorithmization. In fact, the selection of the top level predicate in the graph which is to represent the "central idea" is not easily

mechanizable and is hand-coded although a computer program exists to complete the graph.

Leaving aside text understanding research based on world knowledge schemata - for the reason that we do not propose to use such a scheme, for the time being the only relevant study is the story grammar proposed by Rumelhart.

Rumelhart's work is relevant to the work presented in this paper because it attempts to mechanize the process of textual understanding by encoding the textual structure and world knowledge components into a grammar. The result of this process is a grammar which can be used to "parse" a story structure just as a sentence grammar can be used to parse a sentence. It is our belief that textual grammars are extremely useful in determining the structure and meaning of textual materials for the construction of knowledge bases.

Rumelhart's research has concentrated on story grammars. Our interest lies in the area of technical (expository) text. Rumelhart's grammar, as shown below, depicts a story as consisting, at the highest level, of a setting and an episode.

R1: Story - Setting + Episode

R2: Setting - (State)\*

.\*.

R3: Episode - Event + Reaction

R4: Event - (Episode | Change-of-state | Event + Event)

R5: Reaction ≥ Internal Response + Overt Response

R6: Internal Response ≥ (Emotion | Desire)

R7: Overt Response > (Action | (Attempt)\*

R8: Attempt > Plan + Application

R9: Application > (Preaction)\* + (Action + Consequence)

R10: Preaction > Subgoal + (Attempt)\*

R11: Consequence ≥ (Reaction | Event)

As in any other grammar, the non-terminals Setting and Episode are defined by other productions. A grammar of similar sort could be constructed for an expository text. An example of a grammar for this purpose is shown below.

R1: tutorial > (process-tutorial | device-tutorial }

R2: process-tutorial ≥ introduction + description+ conclusion

R3: device-tutorial > (description)\*

R4: introduction > title

R5: definition > term

R6: process > term

R7: conclusion > term

R8: reason ≥ term

R9: term > (sentence)\* + (description)\*

R10: relation > term

```
Rll: description > {process |
tenn |
definition |
reason |
example |
conclusion }
term + description
```

#### R12: example > term

Comparing the grammars we see some differences. As would be expected, the terms in the expository grammar differ from the terms of the story grammar. Since each textual type serves a different purpose this would be expected. Structurally the grammars also differ. The purpose of an expository text, to explain a concept as phenomena, is reflected in the grammar. A story grammar may include some expository text, but its structure requires additional components for fabricating the story.

#### 3. PROPERTIES OF EXPOSITORY TEXT

Before analyzing a text we should consider some of the properties of expository text which make it expository. Remember, our eventual goal is to be able to automatically analyze textual material. To do so we need to be able to formalize as much about the text as possible.

Kieras [20] notes that one of the distinguishing features of expository text is their noun phrase structure. Specifically, in technical writing, plural nouns seem to be used to refer to concepts and few determiners are used. And there are a lack of use of indefinite noun phrases and when one does occur it usually indicates a new concept being introduced into the text.

Another property of expository text seems to be its inherent hierarchical structure as described earlier. Although named differently by Kieras, three levels of information are discerned; the text grammar; content scheme; content facts. Roughly these correspond to the macrostructural, microstructural, and symbolic levels also described earlier.

#### 4. A SAMPLE TEXT ANALYSIS

This section of the paper makes a preliminary proposal for a descriptive analysis of simple tutorial texts. A selection is made among the various suggestions preserved on the basis of the characteristics of textual material we have chosen and on the ultimate aim we have in mind, namely the automatic generation of a knowledge base for an expert system. Some modifications will be made to adopted models and some new insights will be derived from the experience with our own textual material. The processing level will not be dealt with in this paper, as we believe descriptive modelling should precede process modelling.

The descriptive model has two aspects: (i) what structures should be distinguished at the different levels of descriptive analysis and (ii) how can the discovery of these structures be algorithmatized. The latter aspect has by and large not been considered in the literature.

#### 5. A DESCRIPTIVE MODEL OF TUTORIAL TEXTS

4-3

The texts we have chosen as a basis for our investigation were taken from the encyclopedia for children Why Things Are [1], which aims at explaining elementary natural processes to 8th-9th grade children. This choice was motivated by several considerations, the major one being that these texts presuppose very little prior knowledge, so that we can constrain our analysis to text structure schemata rather than world knowledge schemata. In addition, the texts are short and couched in relatively simple language. Little sophistication about complex text structure is expected from the reader. As an example, to illustrate our analysis let us consider the following text:

- S1) A cloud is made up of tiny droplets of water.
- S2) When a cloud forms the invisible water vapor in the air condenses into visible droplets of water.
- S3) All air contains water.

.\*.

- S4) Warm air can hold more water vapor than cold air.
- S5) If the air cools down it cannot hold so much water vapor and it turns into tiny droplets of water.
- S6) You can see this happen when the hot steam from a kettle cools down in your kitchen.
- S7) A lot of the water vapor turns into a cloud of water droplets.
- S8) Air cools down when it rises because the higher it goes in the atmosphere the cooler it gets.
- S9) In the diagram below the picture on the left shows air rising on a hot day above a city.
- S10) The air rises and cools and the water vapor condenses into droplets to form clouds.
- S11) Towering clouds more than 10 miles high can form in less than an hour on a very hot day.
- S12) Air rising over mountains will cool and clouds will form.
- \$13) Clouds also form when warm moist air rises over a layer of cold air.

On a purely intuitive basis this text seems to be structured on a question-answer pattern, very much in the fashion young children would ask questions of adults, namely "Why is A?", "Because 8", "Why is B", "Because C" and so on...Thus, we might summarize the main gist of the text as follows:

- 0: Why do clouds form?
- Q: What are clouds?
- A: Clouds are made up of water droplets. Clouds form because water vapor in the air condenses into water droplets.
- Q: How does water condense?
- A: Warm air contains more water vapor than cold air. When the air cools down, the water vapor condenses.

- 0: Why does the air cool down?
- A: When the air rises, it cools down because the higher in the atmosphere the cooler. Hencé, the air rises and cools, the water vapor condenses and clouds form.

The following is a more rigorous analysis of the text at the various levels of descriptive analysis. Apart from the 3 levels distinguished above, it seems appropriate to distinguish an additional top-level structure. This top-level subsumes the overall structure of the text. According to Meyer [11] a very common top-level structure is in terms of the rhetorical relation "response" which can take the form of "question answer" or "problem statement - solution". As suggested above, our text is clearly of the form "questionanswer".

The following diagram shows the top-level structure of our text:



it may be noted that the top-level structure corresponds broadly to the paragraph structure of the text.

At the macro-structural level, we analyze each response/answer in terms of its component rhetorical structures.

It would seen that each paragraph (or each level) is composed of the following units:

- definition or statement of fact
- explanation of process or statement of process
- conclusion or summary
- example(s)

The conclusion and the examples are optional but first two units occur universally (except at the top level). Diagrammatically, the following is the structure displayed at the macro-structure level:



The microstructural level deals with the relation between clauses in a sentence and between two successive sentences. As it happens, in our sample text, there are no sentence connectives which denote relations between two sentences, we only have complex sentences containing subordinate conjunctions. We shall ignore the structure of the examples in this initial investigation. Hence, the only complex sentence we need to describe at this level are S2,S5 and S8. At the macro level we classified S2 as being a process statement and S5 and S8 as being process explanations, the latter being more complex than the former. Even though the surface linguistic structures of S5 and S8 are very different, they have the same conceptual structure, which is "fact, BECAUSE process statement", or more explicitly: "A hap-pens, because process B causes A to happen". This structure is explicit in S8 and may be made explicit in S5 if we reformulate it as

"Water vapor turns into tiny droplets of water, because if the air cools down it cannot hold so much water vapor".

Process statements can have different forms: causation in S2 and S5 and sequence in S8.

At the microstructural level we also define the kinds of logical conceptualizations that different clauses may exhibit. The following is a preliminary list:

- definition
- fact
- causation
- comparison
- sequence
- condition
- enumeration

Each of the sentences in our text - excluding the examples - can now be categorized in terms of these concepts:

- S1: definition
- S2: causation S3: definition
- S4: comparison
- S5: fact, because causation S7: fact
- S8: sequence, because causation
- S10: enumeration

The difference between definition and fact is that the former states a general rule about a generic cateogry, whereas the latter states a specific fact. (This distinction being highly relevant for an expert system.)

At the symbolic representation level we propose to represent the sentences by means of "propositions", namely as constructs consisting of a predicate followed by a number of arguments. In doing so, it has to be decided how much decomposition of lexical units should be carried out. For example should the noun phrase "microscopic plant life" be represented as a single constant, such as "microscopic-plant-life" or be decom-posed into its lexical unit by means of a "modifier" predicate, such as (MOD microscopic plant) and (MOD plant life). We have opted for the latter form as it results in greater ability to recognize textual pat-terns. Some guidelines as how to build propositions from sentences are contained in Bovair and Kieras [2], which are broadly followed here. The following show some propositional representations for sample sentences from our text:

- S1: P1 (MADEUPOF) clouds droplets) P2 (HOD droplets water) P3 (MOD droplets tiny)
- S2: P4 (WHEN P5 P6) (FORM clouds) P5 (CONDENSEINTO water-vapor droplets) (MOD water-vapor invisible) P6 P7 **P**8 (IN air water-vapor) **P**9 (MOD droplets water) (MOD droplets visible) P10
- S5: P11 (IF P12 P13) P12 (AND P14 P15) P13 (COOLDOWN air) P14 (NEGATE P16) P15 (CANHOLD air water-vapor) P16 (AMOUNTOF water-vapor less) P17 (TURNSINTO water-vapor droplets) P18 (MOD droplets water)
  - P19 (MOD droplets tiny)

It should be noted that the representation of S5 follows the surface syntactic structure of the sentence and not its reformulized version as suggested above. The microstructure algorithm should be able to recognize this structure as an instance of the appropriate concep-tual category. It should also be noted that pronouns have been replaced by their referents. This again would be done by a special module in the conversion algorithm.

6. ALGORITHMS

6.1 A Concise Description of the Problem

In order to construct a program that will process an expository text and create from it a knowledge base it is necessary to produce all of the structures described in the previous sections of this paper. Having constructed these structures a subsequent processing step would use them to produce the desired knowledge base. The justification for requiring the macrostructure, microstructure, and symbolic level structures is a direct result of our goal to create a knowledge base that is in order to know what the text is about we need the macrostructure. To instantiate causal, problem/ solution, and explanation relations we require the microstructure when these relationships exist between sentences. Finally the symbolic level is needed to instantiate sentence level propositions.

The problem is to transform the raw text into these structures so that the knowledge base can be built. In the next sections we propose how this process can be mechanized.

#### 6.2 Grammars

As shown earlier, grammars give us a way to describe textual structures. We propose a grammar be written for each structural level. Thus, we have a macrostructure grammar, a microstructure grammar, and a symbolic grammar. In addition, we have the standard phrase structure or transformational grammar to produce the sentential constituents. The macrostructure grammar is shown below:

expository text ⇒ collection causation response | comparison description

collection -> (expository text)\*

causation > antecedent + consequent | explanations

response ≥ problem + solution

comparison > alternative adversative analogy

adversative ⇒ favored + unfavored

descriptive > expository text

antecedent ⇒ expository text

consequent -> expository text

explanation ≥ expository text

problem > expository text

solution > expository text

alternative ⇒ expository text

favored  $\ge$  expository text

unfavored > expository text

analogy ≥ expository text

This grammar depicts the highly recursive nature of expository texts - nearly any component can again be a piece of expository text. Having this grammar is not enough since there is no connection between it and the raw structures of a text. In the next section, assuming we have all the necessary grammars we suggest an algorithm for accomplishing the transformation.

6.3 Algorithms for Processing

The following algorithm reprsents the topmost level of processing in our text-to-knowledge base system.

- Create constituent level representation of sentences (phrase structure/transformational grammar)
- Build propositional structures (as described in [2])
- 3) Build microstructures from step 2 structures
- Make macrostructures from step 3 structures
- 5) Use step 2, 3 & 4 structures for building knowledge base.

Needless to say, this algorithm suggests, only in the most abstract way how an expository textual material can be processed. A major part of each step is the pausing process that takes place to produce each step's structures. Each grammar consists of the production rules and a lexicon (specific to the input for the grammar). Both components of the grammar are used in this the recognition process. One assumption of this approach is that all of the categorizations can be determined as a result of surface level features. This may not be the case because some features of expository text are created by inference in the reader. Thus the parsing/recognition mechanism must have the ability to carry out this function.

#### 7. CONCLUSION

In this paper we have barely scratched the surface of what is needed to transform expository text into a knowledge base. We have suggested representations and briefly how they might be created. By using grammars which describe textual structures we can arrive at structures using pausing techniques taht can more readily be processed into a knowledge base. Finally we have suggested what the overall processing scheme is for the program that processes texts. This investigation provides the foundation for our processing program.

#### REFERENCES

- [1] Ardley, N., et al, Why Things Are, Simon & Schuster, New York, 1981.
- [2] Bovain, S., Kieras, D.E., A Guide to Propositional Analysis for Research on Technical Prose, in B.K Britton, J.B. Black, Understanding Expository Text, Lawrence Erlbaum Associates, Hillside, N.J., 1985, pp 315-362.
- [3] Charniak, E., On the use of framed knowledge in language comprehension, Artificial Intelligence, 11, 1978, pp 225-266.
- [4] Charniak, E., Towards a Model of Children's Story Comprehension, Ph.D. Thesis, MIT AI Laboratory, 1972.
- [5] Cowie, J.R., Automatic Analysis of Descriptive Texts, Proceedings, Conference on Applied Natural Language Processing, February, 1983, pp 117-123.
- [6] de Beaugrande, R., Text, Discourse, and Process Toward a Hultidisciplinary Source of Texts, Ablex, Norwood, M.J., 1980.
- [7] Fox, B., Discourse Structure and Anaphora in Written and Conversational English, Ph.D. Thesis, University of California, 1984.
- [8] Grisham, G., Hirschmann, L., Question Answering from Natural Language Medical Data Bases, Volume 7, 1978, pp 333-444.
- [9] Halliday, M.A.K., Hasan, R., Cohesion in English, Longman, London, 1975.
- [10] Kintsch, W., van Dijk, T.A., Toward a Model of Text Comprehension and Production. Psychological Review, 1978, 1985, pp 363-394.
- [11] Meyer, B.J.F., Prose Analysis: Purposes, Procedures and Problems in Understanding, B.K. Britton, J.B. Black, Understanding Expository Text, Lawrence Erlbaum Associates, Hillside, N.J., 1985, pp 11-64.

[12] Meyer, 8.J.F., Freedle, P.O., The Effects of Different Discourse Types of Recall, American Educational Research 1984 Journal, 21, pp 121-143.

....

- [13] Miller, J.R., Kintsch, W., Knowledge-based aspects of prose comprehension and readability of text. 1981, 3, 215-232.
- [14] Minsky, M., A Framework for Representing Knowledge, in The Psychology of Computer Vision, ed. Winston, P.H., McGraw-Hill, New York, pp 211-277.
- [15] Rumelhart, D., Schemata: The Building Blocks of Cognition, In R.J. Sprio, B.C. Bruce, & W.F. Brewer (eds.) Theoretical Issues in Reading Comprehension. Hillsdale, N.J., Lawrence Erlbaum Associates, 1980.
- [16] Rumelhart, D., Notes on a Schema for Stories, in Representation and Understanding, Studies in Cognition, 1975, pp 211-235.
- [17] Schank, R., Lebowitz, M., An Integrated Understander, American Journal of Computation Linguistics, Volume 6, January-March 1980, pp 13-30.
- [18] Schank, R., Abelson, R.P., Scripts, Plans, Goals and Understanding, Erlbaum, Hillside, N.J., 1977.
- [19] Weiner, J.L., Blah, A System Which Explains Its Reasoning, Artificial Intelligence, Volume 15, Number 12, November 1980, pp 19-48.
- [20] Kieras, D.E., Thematic Process in the Comprehension of Technical Prose, in B.K. Britton, J.B. Black, Understanding Expository Text, Lawrence Erlbaum Associates, Hillside, N.J., 1985, pp 89-107.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge Acquisition Workbench for Expert Systems

John H. Boose and Jeffrey M. Bradshaw Knowledge Systems Laboratory, Boeing Advanced Technology Center Boeing Computer Services, P.O. Box 24346, Seattle, WA 98124

# ABSTRACT

Eliciting knowledge from a human expert is a major problem when building a knowledge-based system. The Expertise Transfer System has been expanded into a knowledge acquisition workbench (Aquinas) that combines ideas from psychology and knowledge-based systems research. Aquinas interviews experts and helps them analyze, test, and refine their knowledge. Expertise from multiple experts or other knowledge sources can be represented and used separately or combined. User consultations are run by propagating information through hierarchies. Aquinas delivers knowledge by creating knowledge bases for several different expert system shells. Help is given to the expert by a dialog manager that embodies knowledge acquisition heuristics.

Aquinas contains many techniques and tools for expertise transfer; the techniques combine to make it a powerful testbed for rapidly prototyping portions of many kinds of complex knowledge-based systems.

# **EXPERTISE TRANSFER**

The Expertise Transfer System (ETS) has been in use in Boeing for more than three years. Hundreds of prototypical knowledge-based systems have been generated by ETS. The system interviews experts to uncover vocabulary, conclusions, problem-solving traits, trait structures, trait weights, and inconsistencies. It helps build very rapid prototypes (typically in less than two hours), assists the expert in analyzing the adequacy of the knowledge for solving the problem, and creates knowledge bases for several expert system shells (S.1, M.1, OPS5, KEE, and so on) from its own internal representation (Boose, 1984, 1985, 1986).

ETS has been expanded significantly (Aquinas) to overcome limitations in knowledge representation and reasoning (Figure 1). Due to

#### Features of AQUINAS

- Integrated testing and feedback during the knowledge elicitation process
- Very rapid prototyping of knowledge-based systems
   fast feasibility analysis
  - multiple alternative testing with little resource expenditure
- Case-based elicitation and reasoning
   ease of maintenance
  - comprehensibility
- Tools for comparing knowledge from different experts to show similarities and differences
- Consultation systems giving consensus and dissenting opinions from multiple sources of knowledge
- Multiple methods for dealing with uncertain information
- Significant extensions to personal construct methodology
  - manipulation of rating grids in hierarchies
  - multiple variable scale types
  - interactive testing and debugging of rating grid knowledge
  - many analytic tools in a single framework

Figure 1. Aquinas is a knowledge acquisition workbench that provides a variety of capabilities.

previous limitations, the system was usually abandoned sometime during the knowledge acquisition process. Typically it was used to explore project approaches and assess feasibility for several days or a week, and then development continued in some other expert system shell. While the use of the tool in this way saved substantial time (typically 1 or 2 calendar months from a 12- to 24-month project), it was desirable to make the system more powerful.

# AQUINAS

Aquinas is a collection of integrated tool sets. They share a common user interface (the dialog manager) and underlying knowledge representation and data base (Figure 2). The tools include:

- ETS (Expertise Transfer System) a set of tools used by the expert to elicit, analyze, and refine knowledge as rating grids.
- Hierarchical tools help the expert build, edit, and analyze knowledge hierarchies and lattices. Currently, hierarchies are organized around solutions, traits, knowledge sources (i.e., experts), and cases.
- Uncertain knowledge, preferences, and constraints may be elicited, represented, and applied using combinations of several different methods. Methods are selected based on the cost of elicitation, the precision of the knowledge needed, and convenience.
- A reasoning engine allows consultations to be run on the knowledge in Aquinas. Several

inheritance, specialization, and generalization techniques are employed to propagate knowledge through hierarchies and reach conclusions.

- Various trait (attribute) scale types can be elicited, analyzed, and used by the reasoning engine.
- Several types of tools make inductive generalizations about existing knowledge. Generalizations can be examined by the expert and used to refine the knowledge, and are used by the reasoning engine.
- Knowledge from multiple experts (or other knowledge sources) can be analyzed to find similarities and differences in knowledge, and the degree of subsumption of one expert's knowledge over another. The reasoning engine uses knowledge from user-specified and weighted sources and gives consensus and dissenting opinions.
- A dialog manager makes recommendations to the user about how to use Aquinas.
- Knowledge common to all tool sets is stored in frames in an object-oriented system.

# **Knowledge in Rating Grids**

Aquinas can display part of its problem-solving knowledge as a *rating grid* (Figure 3). Problem solutions - *elements* - are elicited and placed across the grid as column labels, and traits of these solution elements - *constructs* - are listed alongside the rows of the grid. Traits are elicited

Dialog manager									
ETS Repertory grid tools	Hierarchical structure tools	Uncertainty tools	Internal reasoning engine	Multiple scale type tools	Induction tools	Multiple expert tools			
Object-oriented DBMS									
CommonLoops / CommonLisp									

Figure 2. The Aquinas workbench is a collection of integrated tool sets.


Figure 3. Rating values in different hierarchies combine to form *rating grids*. The children of a node in a *solution element hierarchy* supply the elements along the top of the grid; the children of a node in a *trait hierarchy* supply the traits down the side of a grid.

by presenting groups of elements and asking the expert to discriminate among them. Following this, the expert gives each element a *rating* showing where it falls on the trait scale. The interviewing techniques used in building a rating grid are extended from ideas in George Kelly's Personal Construct Psychology (Kelly, 1955) and the PLANET system (Gaines and Shaw, 1981; Shaw and Gaines, 1986). Aquinas can analyze a rating grid in many ways to help the expert refine its problem-solving capability. It uses ratings in the grid to drive user consultations in which the knowledge is tested for necessity and sufficiency.

Single rating grids from Personal Construct Psychology are a fairly weak form of knowledge representation for problem-solving. Although they may be derived quickly, they lack precision and depth:

• A single rating grid can represent only "flat" relations between single elements and traits. No deep knowledge, causal knowledge, or relationship chains can be shown.

- Only elements or traits at the same level of abstraction can be used comfortably in a single grid. Mixing abstraction levels leads to problems during refinement and when the grid is used to make decisions (Boose, 1986).
- Complex problems may be represented only in large grids, which can be difficult for the expert to manipulate and comprehend.
- Hierarchies with multiple experts, several reasoning strategies, and multiple domain models cannot be represented within a single grid.
- It is inconvenient to represent pertain types of problem-solving information solely using Kelly's constructs. Unordered variables, such as a set of computer types, must each be represented as a series of *bipolar* traits (VAX / NOT-VAX, IBM / NOT-IDM...) when it would be easier to to combine them into a single nominal trait (a

COMPUTER trait whose values are VAX, IBM, and so on).

• ETS only reasons with ratings on a scale from 1 to 5, not probabilities or exact numeric values (e.g., dollars or temperature).

#### APPROACH

Aquinas attempts to overcome these limitations as follows:

- By helping experts structure knowledge into solution, trait, expert, and case hierarchies. These hierarchies allow the expert to break up complex problems into chunks of convenient size and similar levels of abstraction.
- By allowing traits to take on unordered or exact numeric rating values when appropriate.

Hierarchical representation and reasoning will be described first, followed by a discussion of trait value types. Then parts of a session typescript for building a Programming Language Advisor will be presented.

#### **Hierarchical Knowledge Representation**

In eliciting knowledge for complex problems it is sometimes difficult for the expert to identify conclusion sets whose members are at similar. useful levels of granularity. For instance, in an engine diagnostic system, the expert may include the repair elements "engine," "battery," "ignition coil," and "electrical system." "Engine" and "electrical system" are at more general levels of structural and functional abstraction than "battery" and "ignition coil." Mixing more general and more specific elements in the same rating grid causes problems during trait elicitation, since traits useful in differentiating "engine" from "electrical system" problems are not necessarily those useful in discriminating "ignition coil" from "battery" problems.

A knowledge acquisition system should allow experts to represent and reason about elements and traits at appropriate levels of generality. Aquinas uses hierarchies to represent rating grids at varying levels of abstraction and to break up problems into subproblems.

Nodes in four hierarchies combine to form rating grids. In the most simple case, the children of a node in a solution hierarchy supply the elements along the top of a grid; the children of a node in a trait hierarchy supply the traits down the side of a grid. Rating values within the grid provide information about the solutions with respect to each trait (Figure 3, taken from the Programming Language Advisor).

Solution hierarchies. Solutions are grouped into specialization hierarchies within Aquinas. This structure aids experts in organizing large numbers of solution elements that may exist at different levels of abstraction. For example, a solution class named "vehicle" is a *superclass* (parent or prototype) to "car" and "truck" *subclasses*. The "car" class can serve in turn as a parent to a class of specific car models or to a particular instance of a car.

Trait hierarchies. Characteristics of a particular level in the solution hierarchy can be structured in trait hierarchies. For instance, in a knowledge base for a Transportation Advisor, the solutions exist in hierarchies of vehicles. Each level in the solution hierarchy has a trait hierarchy that contains information needed to select solutions at that level. A trait hierarchy attached to the "vehicle" abstraction level of a solution hierarchy, for instance, may contain information about general use type, relative speed, cost, and so forth for the types of vehicles in the hierarchy. The "car" subclass is attached to a car trait hierarchy that contains information useful in selecting a particular car.

Two other hierarchies are formed in Aquinas (Figure 4):

Expert hierarchies. Expert hierarchies represent multiple knowledge sources as structured groups. Each node in the expert hierarchy may represent an individual, an aspect of an individual, a group, or an independent knowledge source. Information from multiple experts may be independently elicited and analyzed, then weighted and combined to derive joint solutions to problems. Analyses can be performed that show similarities and differences among experts. Experts each have their own solution and trait hierarchies, which may or may not overlap those of others. An expert's unique problem-solving strategies and information are preserved.

**Case hierarchies.** Case hierarchies define subsets of the knowledge base appropriate to solving a particular class of problems. For example, in a knowledge base of information about vehicles, a user may want to include different knowledge for selecting a vehicle for going over land than for going over water. An *land case* and a *water case* may be created, each drawing on a subset of the expert pool knowledgeable in those areas. Additional levels may be created for short or long land trips, cost considerations, and so on. A hierarchy of cases allows the knowledge base to be developed, modified, and maintained based on specific classes of situations.

A rating grid is built by combining values associated with nodes in each of the four basic hierarchies. Relationships between nodes do not have to be strictly hierarchical; lattices may be formed when more than one parent points to the same child. The expert defines the current rating grid by selecting appropriate nodes in the hierarchies.

Figure 5 shows selected map nodes (case: K-ACQUISITION, expert: WEC, solution: WEC. ELEMENT, trait: WEC. ELEMENT. TRAIT) that describe the rating grid of Figure 3. Each different collection of nodes (at least one from each hierarchy) describes a rating grid. A rating grid could be a single column or row, or even a single cell. Inversely, each cell in a rating grid is uniquely described by its location in the four hierarchies.

In a sense, each rating grid is four dimensional. Any two of these dimensions are shown at once as rows and columns in a given grid. Usually solutions and traits are shown, but sometimes it is useful to show other combinations. For instance, a grid could display the ratings of several experts across the top with particular elements down the side. The associated trait and case nodes would be shown to the side of the grid.

Often the ratings displayed summarize or generalize information from different nodes in the hierarchies; this issue is discussed later.

#### **Reasoning within Knowledge Hierarchies**

The model of problem-solving currently used in Aquinas is that of multiple knowledge sources (experts) that work together in a common problem solving context (case) by selecting the best alternatives for each of a sequential set of decisions (solution elements). Alternatives at each step are selected by combining relevant information about preferences (relativistic reasoning), constraints (absolute reasoning) and evidence (probabilistic reasoning).

For many structured selection problems, a more specialized version of this model seems adequate. After analyzing several expert systems for



Figure 4. Values from expert and case hierarchies as well as solution element and trait hierarchies are combined in many ways to form rating grids. Relationships between nodes do not have to be strictly hierarchical; lattices may be formed when more than one parent points to the same child.



Figure 5. Each cell in a rating grid is described by a unique set of hierarchy nodes. Aquinas users specify rating grids by selecting sets of nodes (either the nodes themselves or their children).

classification, Clancey (1986) suggested that many problems are solved by abstracting data, heuristically mapping higher level problem descriptions onto solution models, and then refining these models until specific solutions are found (Figure 6). This is similar in spirit to the *establish-refine* cycle used in CSRL (Bylander and Mittal, 1986; Chandrasekaran, 1986). In the version of Aquinas described in this paper, data abstraction is carried out within hierarchies of traits, and solutions are refined as information is propagated through solution hierarchies.

Information can be propagated in several ways. In addition to the certainty factor scheme used by ETS, Aquinas makes use of the Analytic Hierarchy Process (AHP; Saaty, 1980) and simple absolute constraint propagation techniques. In the future, probabilistic and userdefined methods will be employed.

## Multiple Rating Value Types

In Personal Construct methodology, ratings are typically represented as *ordinal* values between 1 and 5. Aquinas extends this idea and allows experts to use other types of rating scales. Assigning each trait its own type and range of legal values permits experts to deal with situations where values are unordered or where greater precision is necessary.

In Aquinas, traits are currently described according to the *level of measurement* of their rating scales, which is determined by the expert. The level of measurement depends on the presence or absence of four characteristics: *distinctiveness, ordering in magnitude, equal intervals,* and *absolute zero* (Coombs, Dawes, and Tversky, 1970). These four characteristics describe the four major levels of measurement, or types of traits: *nominal* (unordered), *ordinal, interval,* and *ratio* (Figure 7). The additional information about trait types gives increased power to analytical tools within Aquinas.

Ratings may be generated through several methods:

1. Direct. An expert directly assigns a rating value for a trait and an element. If an exact value is unknown, Aquinas helps the expert derive an estimate (Beyth-Marom & Dekel, 1985). If fine judgments are needed, Aquinas can derive a set of ratio scaled ratings from a series of *pairwise comparisons* (Saaty, 1980). Aquinas also contains tools for encoding of probability distributions on specific values.



Figure 6. Clancey studied structured selection systems and built an abstraction and refinement model (1986). Inference in Aquinas typically occurs in a bottom-up fashion through the trait hierarchies and in top-down fashion through the solution hierarchies.

RATING SCALE	DESCRIPTION	EXAMPLES
Nominal	Unordered set	- LANGUAGE: {ADA COBOL LISP}
Ordinal	Ordered set	- COLD/HOT: {1 2 3 4 5} - SIZE: {SMALL MEDIUM LARGE}
Interval	Ordered set with measurable interva	- SMALL-INTEGERS: als {1234567} - F-TEMP: {32112}
Ratio	Ordered set with measurable interva and an absolute or	-HEIGHT: {0.0' 1.0'} als igin

Figure 7. Aquinas expands the knowledge representation capability of rating grids from Personal Construct Theory by allowing the use of several types of rating scale values. Scale types are selected for convenience, precision, or efficiency of value entry.

2. Derived. Incomplete grids can be automatically filled through propagation of rating values from another grid through the hierarchies (e.g., from lower to higher level grids, different experts, or different cases).

Increased precision and specificity in knowledge acquisition allow increased problem-solving power but usually at some cost (Michalski & Winston, 1985). This cost is reflected in both the *amount of work* needed to elicit the additional information and *increased complexity* and *greater number of steps* in the reasoning process. Aquinas tries to minimize this cost by eliciting more precise information only when it is needed to solve critical portions of the problem. If, for example, Aquinas finds that it cannot sufficiently discriminate between solutions from simple rating values between 1 and 5, it will suggest that the user perform a series of pairwise comparisons to increase the sensitivity of judgments.

#### USING AQUINAS: BUILDING A PROGRAMMING LANGUAGE ADVISOR

Aquinas is written in Interlisp and runs on the Xerox family of Lisp machines. Subsets of Aquinas also run in an Interlisp version on the DEC Vax and a "C/UNIX"-based portable version. The Aquinas screen is divided into a typescript window, map windows showing hierarchies, rating grid windows, and analysis windows (Figure 12). Experts interact with Aquinas by text entry or by mouse through popup menus.

A subsystem called the *dialog manager* contains pragmatic heuristics to guide the expert through knowledge acquisition using Aquinas. Its help is important in the use of Aquinas, given the complexity of the Aquinas environment and the many elicitation and analysis methods available to the expert. The dialog manager makes decisions about general classes of actions and then recommends one or more specific actions providing comments and explanation if desired. This knowledge is contained in rules within the dialog manager in Aquinas. A session history is recorded so that temporal reasoning and learning may be performed (Kitto and Boose, 1986).

Following are the steps in a Aquinas session in which an expert is building a Programming Language Advisor. Novice software engineers and project managers would use such a system to help select programming languages for application projects. Aquinas guides the expert in transferring knowledge into Aquinas's knowledge base, and continues through the making of a knowledge base for the S.1 expert system shell. These steps are:

1. Elicit cases and the initial grid (elements, traits, and ratings). The expert is first asked to specify the behavior of Aquinas's dialog manager. Then the expert enters several problem test cases and selects one for analysis. The knowledge acquisition language case is selected (satellite tracking, accounting, and government transaction cases are also entered). The cases are added to the case hierarchy and appear in the map window (Figure 12; upper right corner). Eventually experts may be able to select and modify grids and cases from a library; we hope that in several years this library will contain hundreds of hierarchies of grids. Building this library bears a modest resemblance to the CYC project at MCC (Lenat, 1986).<sup>1</sup>

The expert chooses to think about a language for developing a knowledge acquisition testbed, and enters potential candidates (Figure 8). After five languages are entered, Aquinas adds an *ideal language* for this problem. This would be an ideal solution for the knowledge acquisition case. The languages are added to the solution hierarchy as children. Then Aquinas asks the expert to enter traits based on differences and similarities between languages. This is the heart of Kelly's interviewing methodology; Aquinas uses it in several different ways as knowledge is expanded through elicitation and analysis.

Aquinas initially assumes that traits will be bipolar with ordinal ratings between 1 and 5. The expert is asked to rate each element with regard to each trait, but the expert may specify different rating scales (unordered, interval, or ratio). Aquinas later assists in recognizing and changing types of rating scales. Aquinas also elicits information about the importance of each trait. This knowledge is used later in the decision making process

2. Analyze and expand the initial, single grid. Once a grid is complete, an analysis is performed to show *implications* between various values of

ELICITING ELEMENTS ---Please enter a list of LANGUAGE elements for K-ACQUISITION, one to a line. When you're done, enter a RETURN. Try to include at least one LANGUAGE that would NOT be good for K-ACQUISITION (a counter example). AQU\*\* ADA AQU\*\* LISP AQU\*\* PROLOG AQU\*\* COBOL AQU\*\* FORTRAN AQU\*\* (CR) Can you imagine a new ideal LANGUAGE for K-ACQUISITION? AQU\*\* YES For now, it will be called IDEAL-K-ACOUISITION-LANGUAGE. -- BUILDING TRAITS BY TRIADS ---When answering the following questions, remember to keep the K-ACQUISITION case in mind. Think of an important attribute that two of ADA, LISP, and PROLOG share, but that the other one does not. What is that attribute? AQU\*\* SYMBOLIC What is that attribute's opposite as it applies in this case? AQU\*\* NUMERIC What is the name of a scale or concept that describes SYMBOLIC/NUMERIC? AQU\*\* ATTITUDE Think of an important trait that two of LISP, PROLOG, and COBOL share, but that the other one does not. What is that trait? AQU\*\* WIDELY AVAILABLE What is that trait's opposite as it applies in this case? AQU\*\* NOT AS WIDELY AVAILABLE What is the name of a scale or concept that describes WIDELY-AVAILABLE/NOT-AS-WIDELY-AVAILABLE? AQU\*\* <u>AVAILABILITY</u> - FILLING IN RATINGS ----Please rate these things on a scale of 5 to 1, where 5 means more like SYMBOLIC and 1 means more like NUMERIC. If neither one seems to apply, enter N(either). If both seem to apply, enter a B(oth). If you would like to change the range or type of scale, enter C(hange scale). SYMBOLIC(5) NUMERIC(1) ADA \*\* 5 LISP \*\* 1 PROLOG \*\* 1 COBOL \*\* 5 FORTRAN \*\* 5 IDEAL-K-ACQUISITION-LANGUAGE \*\* 1

Figure 8. Aquinas asks the expert for an initial set of potential solutions to the first problem case. Then, the solutions are presented in groups of three, and the expert gives discriminating traits. Ratings are entered for each solution for each trait.

<sup>&</sup>lt;sup>1</sup> Lenat is attempting to build a very large knowledge base containing both common sense and expert knowledge to overcome problems of brittleness and labor intensity in designing large software systems.

traits (see the lower right-hand window in Figure 12). Implications are read from left to right, and the thickness of the arc shows the strength of the implication (HARDER. TO. LEARN implies POOR. DEVELOPMENT. ENVIRONMENT). A method similar to ENTAIL (Gaines and Shaw, 1985) derives implications: rating grid entries are used as a sample set and fuzzy set logic is applied to discover inductive implications between the values. This method uncovers higher-order relationships among traits and later helps build trait hierarchies. The expert can also use an interactive process (implication review) to analyze and debug this information: the expert may agree or disagree with each implication. If the expert disagrees, the knowledge that led to the implication is reviewed, and the expert can change the knowledge or add exceptions that disprove the implication (Boose, 1986). Certain types of implication patterns are also uncovered. Discovery of ambiguous patterns, for example, may mean that traits are being used inconsistently (Hinkle, 1965; Boose, 1986).

After the initial grid is complete, the dialog manager suggests a method to help the expert expand the grid. The method depends on the size of the grid, analysis of information in the grid, session history, and so on. The dialog manager inserts the appropriate command on the screen. The expert may change this recommendation or accept it by entering RETURN.

3. Test the knowledge in the single grid. The dialog manager next recommends that the grid knowledge be tested. The expert is asked to provide desirable values for the traits associated with an instance of the case under consideration. These values may be appended with a certainty factor and/or the tag ABSOLUTE to show an absolute constraint. Performance is measured by comparison of experts' expectations with Aquinas consultation results.

Two methods are available in Aquinas for turning rating values in grids into solution recommendations. One approach involves mapping this information onto certainty factor scales. Each rating in the grid is assigned a certainty factor weight based on its *relative strength* (a 5 is stronger than a 4), the *relative weight* the expert has assigned to the trait, and any *absolute constraints* that the expert has specified for the trait. In the test consultation, EMYCIN's certainty factor combination method combines the certainty factors in the grid (Adams, 1985). The result is a rank-ordered list of solutions with certainty factor assignments. These certainty factors are also used when rules are generated for expert system shells.

Another approach used by Aquinas to handle uncertainty employs Saaty's Analytic Hierarchy Process to order a set of possible solutions. Grid information obtained through pairwise comparisons or through regular rating grid methods is mapped onto *judgment matrices*. The *principal eigenvector* is computed for each matrix; the eigenvectors are normalized and combined to yield a final ranking of the solutions. Each solution has a score between 0.0 and 1.0. In a knowledge base consisting of multiple grids, these values are propagated through the hierarchies.

4. Build hierarchies (structured as elements and traits in multiple grids) from the first grid. Next, the dialog manager recommends that the expert expand the trait and solution hierarchies by performing a *cluster analysis* (Figure 9). Aquinas uses a method of single-link hierarchical cluster analysis based on FOCUS (Shaw and Gaines, 1986) to group sets of related solutions or traits. The junctions in the clusters can be seen as conjectures about possible new classes of solutions or traits. These more general trait or solution classes may be named and added to the hierarchies.



Figure 9. Solution element and trait clusters are formed from information in rating grids. The expert is asked to label nodes and expand clusters; new traits are used to expand the hierarchies.

Laddering is also used to find traits at varying levels of abstraction (Boose, 1986). "Why?" questions are used to find more general traits:

What is a new trait that says why you think GOOD-DEVELOPMENT-ENVIRONMENT should be true of a LANGUAGE for K-ACQUISITION?

AQU\*\* FASTER SYSTEM DEVELOPMENT

"How?" questions help find more specific traits:

In what ways could a language for K-ACQUISITION be characterized by WIDELY-AVAILABLE?

AQU\*\* RUNS ON MULTIPLE HARDWARE AQU\*\* MANY COMPILERS AVAILABLE AQU\*\* MANY COMPANIES OFFER

Generally, experts stop expanding the trait hierarchies when they are able to provide direct grid ratings at these more specific trait levels. Ratings need not be explicitly given at each level of the trait and solution hierarchies, but can often be inferred form other grids in the knowledge base (e.g., induction from more specific examples or inheritance from more general ones) (Lieberman, 1986).

5. Use several rating value types (transform ordinal ratings to nominal and interval ratings) to represent knowledge. Aquinas helps the expert convert a trait with ordinal values (DELIVERY-COST: HIGH-COST(5) / LOW-COST(1)) into a trait with ratio scaled rating values (DELIVERY-COST: (1500 - 60000) DOLLARS-US). The expert re-rates the solutions in terms of the new values and these values appear on the grid. Aquinas provides several forms of estimation help. Four estimation procedures are provided: START-&-MODIFY, EXTREME-VALUES, DECOMPOSITION, and **RECOMPOSITION** (Beyth-Marom & Dekel, 1985). In this instance, the EXTREME-VALUES procedure first asks for the least and greatest DELIVERY-COST one could imagine for the type of Lisp being considered. Through a series of questions, Aquinas helps shrink this range until a satisfactory estimate is given.

Aquinas also helps the expert change trait scale types by checking values associated with particular kinds of traits. For instance, bipolar traits that receive only extreme ratings (e.g. RUNS. ON. VAX/RUNS. ON. IBM) may be better represented with an unordered trait (e.g. COMPUTER. TYPE). 6. Test knowledge in hierarchies; test knowledge from multiple experts. Another expert adds knowledge about programming language selection to the knowledge base and tests it. In the first consultation (Figure 10), the user is interested in selecting a particular version of Lisp, Prolog, or ADA for a knowledge acquisition project. Because of the many potential solutions, the user is given the opportunity to specify a subset for consideration. The solutions in this subset are called *solution candidates*.

Aquinas then asks for a set of trait values for this consultation. The user enters the absolute constraint that only languages with a delivery cost of less than \$30,000 will be considered. The user may accept default values entered in a previous consultation by pressing the RETURN key. If a default value has not been previously

-- TEST CONSULTATION --Would you like to run an EXISTING or NEW consultation? AQU\*\* EXISTING What is the name of this existing consultation? AQU\*\* LISP-PROLOG-ADA-ONLY This test consultation is named K-ACOUISITION.LISP-PROLOG-ADA-ONLY. Which K-ACQUISITION alternatives you would like to consider in this consultation (LISP-PROLOG-ADA-ONLY). Enter them one to a line. If you wish all solutions to be considered, type ALL. When done, press RETURN. AQU\*\* MACLISP-LM AQU\*\* INTERLISP AQU\*\* <u>COMMON-LISP</u> AQU\*\* <u>QUINTUS-PROLOG</u> AQU\*\* <u>ADA-1</u> AQU\*\* \*\* (<u>CR</u>) The following experts know about MACLISP-LM, INTERLISP, COMMON-LISP, QUINTUS-PROLOG, and ADA-1: WEC JCA. Would you like to exclude or weight any of these experts? AQU\*\* NO Please indicate the desired trait selection values for LISP-PROLOG-ADA-ONLY solutions. Press RETURN to indicate agreement with the default values, or type in a new value. Values may be appended with a certainty factor in the form '.8' and/or the word ABSOLUTE to indicate that the value is an absolute constraint when selecting a type of LANGUAGE for K-ACQUISITION. (WIDELY-AVAILABLE(5), 1.0)\*\* (CR) (GOOD-DEVELOPMENT-ENVIRONMENT(5), 1 0)\*\* (CR) (LOW-COST(<45000 DOLLARS-US), 1.0, ABSOLUTE) (NOTE: THIS INCLUDES HARDWARE FOR A WORKSTATION) \*\* <30000 DOLLARS-US 1.0 ABSOLUTE

Figure 10. The expert tests the knowledge by running a consultation. The expertise of two experts is used and consensus and dissenting solutions are given (see Figure 11).

specified and the user types RETURN, that trait will be ignored in the inference process for this consultation. The user's preference for HARDWARE type is partitioned among three manufacturers by pairwise comparison (Figure 11), which generates a ratio scaled set of preferences (Saaty, 1980).

The results of the consultation are presented to the user. For each solution, the *consensus* recommendation of the experts consulted is presented, followed by the weight of each expert that contributed to the recommendation. With multiple experts, it may sometimes be useful to

(COMPANIES(VAX .33, IBM .33, ATT .33), 1.0)\*\* PAIRWISE Please compare these values of HARDWARE with regard to their importance in contributing to an overall high score for a particular type of LANGUAGE for K-ACQUISITION in the context of LISP-PROLOG-ADA-ONLY. Please compare VAX and IBM. Enter: VAX = IBM if VAX and IBM are equally important VAX>IBM or VAX<IBM if one of the pair is weakly more important VAX>>IBM or VAX<<IBM if one is strongly more important VAX>>>IBM or VAX<<<IBM if one is demonstrably or very strongly more important VAX>>>>IBM or VAX<<<<IBM if one is absolutely more important AQU\*\* VAX <IBM Please compare VAX and ATT. Enter: VAX = ATT if VAX and ATT are equally important VAX>ATT or VAX<ATT if one of the pair is weakly more important VAX>>ATT or VAX<<ATT if one is strongly more important VAX>>>ATT or VAX<<<ATT if one is demonstrably or very strongly more important VAX>>>>ATT or VAX<<<<ATT if one is absolutely more important AQU\*\* VAX>>>>ATT Results for test consultation K-ACQUISITION.LISP-PROLOG-ADA-ONLY: 1: INTERLISP (.47: (WEC .5, JCA .5)) 2: QUINTUS-PROLOG (.40: (WEC 1.0)) Would you like to see the dissenting opinion for this consultation?

AQU\*\* <u>YES</u>

The following dissenting opinion was given by WEC: Overall agreement with consensus: .79 1: QUINTUS PROLOG (.40) 2: INTERLISP (.39)

Figure 11. Test consultation (continued). The expert specifies "run-time" values for traits, entering an absolute cost constraint, and performing a pairwise comparison task to derive relative values for hardware.

examine a set of recommendations from a dissenting expert or group of experts. Since WEC's recommendations differed most from the consensus, these are listed as a dissenting opinion.

A general model illustrating the inference propagation path was shown in Figure 6. For each expert consulted and for each level in that expert's solution hierarchy, a partial problem model is constructed, evaluated, and abstracted in a bottom-up fashion through the trait hierarchy of that solution level. Through this process the solution is refined as the children of the best solutions are chosen for continued evaluation. Bottom-up abstraction takes place again in the trait hierarchy at the new solution level, and the cycle continues until all remaining solution candidates have been evaluated. Then an ordered list of solution candidates is obtained and combined with the results from other experts. This information from a single case may then be combined, if desired, with information from other cases to derive a final ranking of solution candidates. Users may override this general model of inference propagation by specifying explicit inference paths and parameters.

Selection of the best solution at a given level requires that Aquinas use two main types of inference through the associated trait hierarchy: absolute and relativistic.

Absolute reasoning. Absolute reasoning involves judgments made with no significant reservations. It "typically depends on relatively few facts, its appropriateness is easy to judge, and its result is unambiguous" (Szolovitz and Pauker, 1978). For example, in selecting a programming language, users may be able to say with certainty that they would be interested only in languages that run on an Apple Macintosh or that they will not consider a language that costs more than \$400, regardless of other desirable characteristics. Experts can also build these types of absolute constraints into an Aquinas rating grid. Absolute reasoning in Aquinas is somewhat similar to solving a set of linked decision tables (Hurley, 1983; Michalski, 1978). Some of the inductive generalization rules described by Michalski (1983) prove useful in this context.

**Relativistic reasoning.** Unfortunately, not all judgments can be absolute. Many involve

significant trade-offs, where information and preferences from several sources must be weighed. Even if criteria for the ideal decision can be agreed on, sometimes it can be only approximated by the available alternatives. In these cases, problem-solving information must be propagated in a relativistic fashion. Aquinas incorporates a variety of models and approaches to relativistic reasoning, including MYCIN-like certainty factor calculus (Adams, 1985), fuzzy logic (Gaines and Shaw, 1985), and the Analytic Hierarchy Process (AHP, Saaty, 1980).

Probabilistic and user-defined reasoning. Future versions of Aquinas will have models for the elicitation (Alpert and Raiffa, 1982; Spetzler and von Holstein, 1984; Wallsten and Budescu, 1983) and analysis of probabilistic information including Bayesian (Howard and Matheson, 1981; Cheeseman, 1985; Henrion, 1986: Pearl 1986; Spiegelhalter, 1986), Dempster-Shafer (Shafer, 1976; Gordon and Shortliffe, 1985), and other related approaches approaches (Shastri and Feldman, 1985). We also plan to allow userdefined methods for combining and propagating information.

The availability of different inference methods within a single workbench allows users and experts flexibility in adapting Aquinas to the problem at hand. Most knowledge engineering tools do not allow experts to specify how information should be combined. They tend to use fixed, global numeric functions to compute values. Future research will suggest heuristics for selecting appropriate methods and designs for particular types of questions (e.g., Shafer and Tversky, 1985). These heuristics will be incorporated into Aquinas's dialog manager.

7. Edit, analyze, and refine the knowledge base, building new cases. Once the experts have entered information about one case, they describe additional cases. They could start from scratch by entering a list of relevant solutions and traits, but that would be inefficient if there were significant overlap in those required by a previously entered case and a new one (Mittal, Bobrow, and Kahn, 1986). Aquinas allows an expert to copy pieces of hierarchies (and, optionally, their associated values) between cases. Information copied in this way can be modified to fit the new context. This facility may also be used to copy pieces of hierarchies between experts. 8. Generate rules for expert system shells. The expert is the judge of when the point of diminishing returns has been reached within Aquinas. When such a point is reached, a knowledge base is generated for an expert system shell, and development continues directly in that shell. Similarity and implication analyses allow experts to determine whether traits or solutions can be adequately and appropriately discriminated from one another. The system provides correlational methods for comparing the order of Aquinas recommendations to an expert's rankings.

Aquinas can generate knowledge bases for several expert system shells. The knowledge contained in grids and hierarchies is converted within Aquinas into rules, and the rules are formatted for a particular expert system shell. The appropriate control knowledge is also generated. Rules are generated with screening clauses that partition the rules into subsets. An *expert clause* is used when expertise from multiple experts is weighted and combined together. A *case clause* controls the focus of the system during reasoning.

Four types of rules are generated:

- 1. Implication rules are generated from arcs in the implication graph and conclude about particular traits. The conclusion's certainty factor is proportional to the strength of the implication. The use of implication rules restricts search and lessens the number of questions asked of users during consultations.
- 2. Solution rules conclude about a particular solution or solution class. The conclusion's certainty factor is derived from a combination of the grid rating strength and the trait weight.
- 3. Absolute rules are generated when the expert places an absolute constraint on the value of a trait. Sometimes information about absolute constraints is included elsewhere when knowledge bases for expert system shells are generated.
- 4. Specialization / Generalization rules are derived from information in the hierarchies and are used to propagate hierarchical information.

9. Further expand and refine the knowledge base. Hierarchies and rating grids continue to be used during the session to expand and refine the knowledge base. Work in progress is shown in Figure 12. Aquinas contains a variety of other tools to help analyze and expand the knowledge base:

**Comparison of Experts (Sources).** The MINUS tool (Shaw and Gaines, 1986) compares grids from different experts on the same subject and points out differences and similarities. This information has been used to manage structured negotiation between experts (Boose, 1986). SOCIOGRIDS features (Shaw and Gaines, 1986) will be available in the future to display *networks* of expertise. Nodes and relations in these networks show the degree of subsumption of one expert's grid over grids from other experts.

Incremental Interviewing. Aquinas can use an incremental dialog to elicit new traits and solutions, one at a time, from the expert (Boose, 1986). This is useful when the expert does not have a list of elements to start a grid and in other situations during knowledge refinement.

Trait Value Examination. New elements can be identified by asking the expert to "fill in holes" in the values of trait ranges. For instance, no element may exist with a rating of 2 on some ordinal trait scale; the expert is asked if such an element can be identified:

What is a new LANGUAGE that would receive a value of 2 on the scale SCIENTIFIC(5) / BUSINESS(1)?

New traits can also be identified by forming triads based on ratings: if LISP and PROLOG are rated 5 on SCIENTIFIC(5) / BUSINESS(1), and ADA is rated 4, the expert is asked,

What is a new trait having to do with SCIENTIFIC / BUSINESS that makes LISP and PROLOG similar yet different from ADA?

## Trait Range Boundary Examination.

Important traits can frequently be identified by exploring the boundaries of trait ranges:

You said that the range of DELIVERY-COST for LISP for the K-ACQUISITION case was 1500 to 60000 DOLLARS-US. Can you think of any conditions in the future that might make DELIVERY-COST LESS THAN 1500?

#### AQU\*\* YES

:

Enter conditions in terms of traits, one to a line; enter a

RETURN when done.

AQU\*\* HARDWARE BREAKTHROUGH - LISP ON A CHIP AQU\*\* (CR)

Can you think of any conditions in the future that might make DELIVERY-COST GREATER THAN 60000? AQU\*\* <u>YES</u>

Enter conditions in terms of traits, one to a line; enter a RETURN when done.

AQU\*\* <u>VERY POWERFUL HARDWARE</u> AQU\*\* <u>PARALLEL ARCHITECTURES AVAILABLE</u> AQU\*\* (<u>CR)</u>

**Completeness Checking.** A single grid can be used as a table of examples. If the table is incomplete, the expert is asked to fill in other examples.

**Combine Similar Traits.** Sometimes different labels are used for the same underlying concept. This can be discovered when a similarity analysis is performed (functionally equivalent traits with different labels may be uncovered). If the expert cannot think of a new element to separate identical traits, then the traits may be combined into a single trait.

#### DISCUSSION

# General Advantages and Disadvantages of Aquinas

Aquinas inherits the advantages of ETS: rapid prototyping and feasibility analysis, vocabulary and trait elicitation, testing and refinement tools, implication discovery, conflict point identification, expert system shell production, and generation of expert enthusiasm (Boose, 1986).

Over 30 prototype systems have been built during the development of Aquinas (an AI Book Consultant, an AI Tool Advisor, a Course Evaluation System, a Customer Needs Advisor, a Database Management System Consultant, an Investment Advisor, a Management Motivation Analyzer, a Personal Computer Advisor, a Personality Disorder Advisor, a Product Design and Impact Advisor, a Robotic Tool Selector, a Seattle Travel Agent, and a Wine Advisor, among others). The Programming Language Advisor session took less then 2 hours with each of the two experts.

Aquinas offers a rich knowledge representation and reasoning environment. We believe that Aquinas can be used to acquire knowledge for



Figure 12. Aquinas screen showing developing hierarchies, a rating grid, and an implication analysis graph of the grid.

significant portions of most structured selection expert system problems. Hierarchies help the expert break down problems into component parts and allow reasoning at different levels of abstraction. Varying levels of precision are specified, with multiple types of rating scales when needed.

Knowledge from multiple experts may be combined using Aquinas. Users may receive dissenting as well as consensus opinions from groups of experts, thus getting a full range of possible solutions. Disagreement between the consensus and the dissenting opinion can be measured to derive a degree of conflict for a particular consultation. The system can be used for cost-effective group data gathering (Boose, 1987). The growing collection of rating grids and case knowledge represents an important resource for building a variety of knowledge-based systems. Knowledge is stored explicitly with associated problem cases, making knowledge bases easier to update and maintain.

Currently, a user may copy and change any portion of the Aquinas knowledge base during a consultation. In the future, each expert will be able to protect areas of knowledge. The expert may believe protection is necessary because some knowledge should not be changed or because the knowledge has commercial value.

Aquinas is not as easy to use as was ETS using single grids. There are many elicitation and analysis tools for a novice to understand; the decision-making process and inference engine can be set up to work in several different ways. We expect that continuing improvements in the dialog manager will help make the system more comprehensible and decrease the learning time for new users.

#### **Theoretical Issues - Knowledge Elicitation**

Personal Construct Psychology methods provide no guarantee that a *sufficient* set of knowledge will be found to solve a given problem. Aquinas attempts to expand the initial subset of solutions and traits based on problem-solving knowledge for specific cases. The goal is to solve enough cases so that the knowledge is sufficient to solve *new* cases. This is the methodology of knowledge engineering in general; Aquinas helps make the process explicit and manageable.

Hierarchical decomposition can be used to build intuitive, comprehensible models that seem to behave in reasonable ways. One disadvantage is that some problems do not easily fit the hierarchical model. It also may be true that a particular problem would best be represented by a collection of conflicting hierarchies (hierarchies for mechanical problems tend to model structure or function, not both, and both may be necessary).

The use of multiple rating value types provides more flexibility, convenience, and precision in representing knowledge. However, deciding a particular type of variable to use can be a complex task. The dialog manager offers some assistance, but the expert usually must learn appropriate usage of rating types through experience.

Experts develop Aquinas knowledge bases serially. In the future, we would like to build a participant system in which many experts could dynamically share rating grids and hierarchies (Chang, 1985).

#### Analysis and Inference

Multiple analysis tools and elicitation methods in Aquinas help the expert think about the problem in new ways and tend to point out conflicts and inconsistencies over time. Lenat argues that knowledge representations should shift as different needs arise (Lenat, 1983). This should lead to better problem and solution descriptions, and, in turn, to better problemsolving. Inference in Aquinas is efficient because the problem space is partitioned. Information in the trait hierarchies is attached to particular levels of solution elements. Although no formal studies have been conducted, consultation results using the methods described above seem reasonable.

Rule generation for expert system shells is straightforward. Development of the knowledge base can continue in an expert system shell that may offer advantages of speed, specialized development and debugging facilities, and inexpensive hardware.

#### **Future Directions**

We intend to build a knowledge acquisition environment that includes specific domain knowledge for specialized application areas and can acquire knowledge for synthetic problems, combining features from other knowledge acquisition tools such as MDIS (Antonelli, 1983), DSPL (Brown, 1984), MORE (Kahn, Nowlan, and McDermott, 1985), and PRIDE (Mittal, Dym, and Morjaria, 1986).

Presently Aquinas works best on those problems whose solutions can be comfortably enumerated (analytic or structured selection problems such as classification or diagnosis) as opposed to problems whose solutions are built up from components (synthetic or constructive problems such as configuration or planning). Simple classification can be thought of as a single decision problem (handled by ETS). Complex structured selection problems may require a set of linked data abstraction/solution refinement decisions (Aquinas). The next step may be to generalize this process to acquire and represent knowledge for planning, configuration, and design problems where the order of linked decisions in solution hierarchies may represent precedence of events or goals rather than just solution refinement. In these problems hierarchies may be assembled at consultation time rather than constructed totally in advance as they are currently. Grid cells might sometimes contain an arbitrary computation rather than a rating. These would include results of functions (such as found in spreadsheets) or database retrievals. Deeper models of the structure and function of physical systems could be modeled.

An important step in expanding the knowledge acquisition workbench concept is the linking

together of other specialized tools. At the Boeing Knowledge Systems Laboratory we are investigating ways of integrating diverse knowledge representations from different Laboratory projects so that this may be more easily accomplished. In the domain of knowledge acquisition, we feel that the approach used in SALT (Marcus, McDermott, and Wang, 1985; Marcus and McDermott, 1986) is particularly promising. SALT is a system that interviews experts to build knowledge bases for certain types of constructive problems (its first use was to configure elevators). We are also interested in generating knowledge sources for BBB, a blackboard system that has been successfully applied to a variety of problems (Benda et al., 1986).

Development of the Aquinas workbench will continue in an incremental fashion. Techniques will be continuously integrated and refined to build an increasingly more effective knowledge acquisition environment.

#### ACKNOWLEDGEMENTS

Thanks to Roger Beeman, Miroslav Benda, Kathleen Bradshaw, William Clancey, Brian Gaines, Cathy Kitto, Ted Kitzmiller, Art Nagai, Doug Schuler, Mildred Shaw, David Shema, Lisle Tinglof-Boose, and Bruce Wilson for their contributions and support. Aquinas was developed at the Boeing Artificial Intelligence Center of Boeing Computer Services in Seattle, Washington.

#### BIBLIOGRAPHY

Adams, J., Probabilistic Reasoning and Certainty Factors, in B. Buchanan and E. Shortliffe (eds.), <u>Rule-Based Expert Systems:</u> <u>The MYCIN Experiments of the Stanford</u> <u>Heuristic Programming Project</u>, Reading, MA: Addison-Wesley, 1985.

Alpert, M., and Raiffa, H., A Progress Report on the Training of Probability Assessors, in D. Kahneman, P. Slovic, and A. Tversky (eds.), <u>Judgment Under Uncertainty: Heuristics and</u> <u>Biases</u>, New York: Cambridge University Press, 1982.

Antonelli, D., The Application of Artificial Intelligence to a Maintenance and Diagnostic Information System (MDIS), in the Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance, Boulder, CO., 1983. Benda, M., Baum, L. S., Dodhiawala, R. T., and

Jagannathan, V., Boeing Blackboard System, in the Proceedings of the High-Level Tools Workshop, Ohio State University, October, 1986. Beyth-Marom, R., and Dekel, S., <u>An Elementary</u> <u>Approach to Thinking Under Uncertainty</u>, London: L. Erlbaum Assoc., 1985.

Boose, J. H., Personal Construct Theory and the Transfer of Human Expertise, in the Proceedings of the National Conference on Artificial Intelligence, Austin, Texas, 1984.

Boose, J H., A Knowledge Acquisition Program for Expert Systems Based On Personal Construct Psychology, Int. J. Man-Machine Studies, 23, 1985.

Boose, J H., <u>Expertise Transfer for Expert</u> <u>System Design</u>, New York: Elsevier, 1986.

Boose, J. H., Rapid Acquisition and Combination of Knowledge from Multiple Experts in the Same Domain, *Future Computing Systems Journal*, in press, 1987.

Brown, D. E., Expert Systems for Design Problem-Solving Using Design Refinement with Plan Selection and Redesign, unpublished Ph.D. dissertation, Ohio State University, CIS Department, Columbus, Ohio, Aug., 1984.

Bylander, T., and Mittal, S., CSRL: A Language for Classificatory Problem-Solving and Uncertainty Handling, *AI Magazine*, Aug., 1986.

Chandrasekaran, B., Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design, *IEEE Expert*, Fall, 1986.

Chang, E., Participant Systems, Calgary, Alberta: Alberta Research Council Advanced Technologies, Unpublished manuscript, December 1985.

Cheeseman, P., In defense of probability, in the Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, California, 1985.

Clancey, W., Heuristic Classification, in J. Kowalik (ed.), <u>Knowledge-Based Problem-</u> <u>Solving</u>, New York: Prentice-Hall, 1986.

Coombs, C. H., Dawes, R. M., and Tversky, A., <u>Mathematical Psychology</u>, Englewood Cliffs, NJ: Prentice-Hall, 1970.

Gaines, B. R., and Shaw, M. L. G., New Directions in the Analysis and Interactive Elicitation of Personal Construct Systems, in M. L. G. Shaw (ed.), <u>Recent Advances in Personal</u> <u>Construct Technology</u>, New York: Academic Press, 1981.

Gaines, B. R., and Shaw, M. L. G., Induction of Inference Rules for Expert Systems, *Fuzzy Sets* and Systems, August, 1985.

Gordon, J. and Shortliffe, E., The Dempster-Shafer Theory of Evidence, in B. Buchanan and E. Shortliffe (eds.), <u>Rule-Based Expert Systems:</u> <u>The MYCIN Experiments of the Stanford</u> <u>Heuristic Programming Project</u>, Reading, MA: Addison-Wesley, 1985.

Henrion, M., Propagating Uncertainty by Logic Sampling in Bayes' Networks, in the *Proceedings*  of the Second Workshop on Uncertainty in Artificial Intelligence, Philadelphia, PA, 1986.

Hinkle, D. N., The Change of Personal Constructs from the Viewpoint of a Theory of Implications, Ph. D. dissertation, Ohio State University, OH, 1965.

Howard, R. A. and Matheson, J. E., Influence Diagrams, in R. A. Howard and J. E. Matheson (Eds.), <u>Readings on the Principles and</u> <u>Applications of Decision Analysis</u>, Menlo Park, California: Strategic Decisions Group, 1984.

Hurley, R., <u>Decision Tables in Software</u> <u>Engineering</u>, New York: Van Nostrand Reinhold, 1983.

Kahn, G., Nowlan, S., and McDermott, J., MORE: An Intelligent Knowledge Acquisition Tool, in the *Proceedings of the Ninth Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, 1985.

Kelly, G. A., <u>The Psychology of Personal</u> <u>Constructs</u>, New York: Norton. 1955.

Kitto, C., and Boose, J. H., Heuristics for Expertise Transfer: The Automatic Management of Complex Knowledge Acquisition Dialogs, *Proceedings of the IEEE Expert Systems in Government Conference*, McClean, VA, October, 1986.

Lenat, D., The Nature of Heuristics, Artificial Intelligence, 19 (1983) and 21 (1983).

Lenat, D., Prakash, M., and Shepard, M., CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks, *The AI Magazine*, **6** (4), 1986.

Lieberman, H., Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems, in the Proceedings of the Object-Oriented Programming Systems, Languages, and Applications Workshop, Portland, Oregon (and special issue of SIGPLAN Notices, Nov., 1986).

Marcus, S., McDermott, J., and Wang, T., Knowledge Acquisition for Constructive Systems, in the *Proceedings of the Ninth Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, 1985.

Marcus, S., and McDermott, J., SALT: A Knowledge Acquisition Tool for Propose-and-Revise Systems, Carnegie-Mellon University Department of Computer Science technical report, forthcoming.

Michalski, R. S., Designing Extended Entry Decision Tables and Optimal Decision Trees Using Decision Diagrams, Urbana, IL.: Intelligent Systems Group, Artificial Intelligence Laboratory, Department of Computer Science, University of Illinois, 1978.

Michalski, R. S., A theory and Methodology of Inductive Learning, in R. Michalski, J. Carbonell, and T. Mitchell (eds.) <u>Machine</u> <u>Learning</u>, Palo Alto, CA: Tioga Publishing, 1983. Michalski, R., and Winston, P., Variable Precision Logic, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, MIT AI Memo 857, 1985.

Mittal, S., Bobrow, D., and Kahn, K., Virtual Copies: At the Boundary Between Classes and Instances, in the Proceedings of the Object-Oriented Programming Systems, Languages, and Applications Workshop, Portland, Oregon (and special issue of SIGGPLAN Notices, Nov., 1986).

Pearl, J., Fusion, Propagation and Structuring in Belief Networks, technical report CSD-850022, R-42-VI-12, Cognitive Systems Laboratory, Computer Science Department, University of California, Los Angeles, CA, April 1986.

Saaty, T. L., <u>The Analytic Hierarchy Process</u>, New York: McGraw-Hill, 1980.

Shafer, G., and Tversky, A., Languages and Designs for Probability Judgment, *Cognitive Science*, 9, 1985, 309-339.

Shastri. L., and Feldman, J., Evidential Reasoning in Semantic Networks: A Formal Theory, in the Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, CA, August, 1985.

Shaw, M. L. G., and Gaines, B. R., PLANET: A Computer-Based System for Personal Learning, Analysis, Negotiation and Elicitation Techniques, in J. C. Mancuso and M. L. G. Shaw (eds.), <u>Cognition and Personal Structure:</u>

Computer Access and Analysis, Praeger Press, 1986, in press.

Slater, P., <u>Dimensions of Interpersonal Space</u>, <u>Vol. 2</u>, London: Wiley and Sons, 1977.

Spetzler, C., and Stal von Holstein, C., Probability Encoding in Decision Analysis, in R. Howard and J. Matheson (eds.), <u>Readings on the</u> <u>Principles and Applications of Decision Analysis</u>, Volume 2, Palo Alto, CA: Strategic Decisions Group, 1983.

Spiegelhalter, D. J., Probabilistic Reasoning in Predictive Expert Systems, in L. N. Kanal and J. Lemmer (Eds.), <u>Uncertaintv in Artificial</u> <u>Intelligence</u>, Amsterdam: North-Holland, 1986.

Szolovits, P., and Pauker, S., Categorical and Probabilistic Reasoning in Medical Diagnosis, *Artificial Intelligence*, 11, 1978.

Wallsten, T., and Budescu, D., Encoding Subject Probabilities: A Psychological and Psychometric Review, *Management Science*, **29**:2, 1983.

# INDUCTION OF HORN CLAUSES: METHODS AND THE PLAUSIBLE GENERALISATION ALGORITHM

Wray Buntine New South Wales Inst. of Tech. and Macquarie University

> Computing Science N.S.W.I.T. P.O. Box 123, Broadway, 2007 Australia

**Abstract.** We are considering the problem of induction of uncertaintyfree descriptions for concepts when arbitrary background knowledge is available, to perform constructive induction for instance. As an idealised context, we consider that descriptions and rules in the knowledge-base are in the form of Horn clauses. Using a recently developed model of generality for Horn clauses, we argue that some induction techniques are inadequate for the problem. We propose a framework were induction is viewed as a process of model-directed discovery of consistent patterns (constraints and rules) in data, and describe a new algorithm, the plausible generalisation algorithm, that tackles the sub-problem of discovering rules. The algorithm raises a number of interesting questions: How can we identify and constrain *irrelevance* during the generalisation process? How can our knowledge-base answer queries of the form: "what do (objects) X and Y have in common that is relevant to (situation) S?"

# 1. INTRODUCTION

A key technique for use in any semi-automatic acquisition system for knowledge-based systems is concept induction (Michalski, 1983; Michie, 1986; Quinlan, 1986a). By concept induction, we mean the discovery of a description for a concept when only positive and negative examples of the concept are known (Dietterich et al., 1982; Rendell, 1986). In practice, such induction may proceed in the presence of a rich body of background knowledge, for instance allowing new descriptors to be derived about examples. So constructive induction, induction incorporating the derivation of new descriptors (Michalski, 1983), may well be required.

In this paper we are considering the induction of logical, uncertainty-free descriptions (*logic-based induction*) in the presence of logical background knowledge, and consequently, constructive induction. Our treatment is incomplete, however, as it makes only passing mention of the roles that *credibility of hypotheses* and *noise* have to play in logic-based induction or of the *heuristic search* of a space of hypotheses (see for example Quinlan, 1986b; Cheeseman, 1984, 1986; Rendell, 1986).

We consider the idealised context where the concept is to be described by Horn clauses (the language on which Prolog is based; see for example Robinson, 1979; Lloyd, 1984) and arbitrary background knowledge has been expressed in Horn clause form. Arguments supporting the use of Horn clauses to pursue logic-based induction are given in the next subsection. Additionally, we believe that by having a good understanding of inductive learning and its applications in the context of Horn clauses, we can improve our understanding of comparable problems in more extensive knowledge representations, for instance, ones incorporating uncertainty with deduction or using a more complete subset of 1st-order logic.

In the third section we review a number of existing AI logic-based induction methods, showing they are inadequate for the more extensive context considered here, that is incorporating background knowledge/constructive induction. This analysis is made on the basis of a recently devised model of generality for Horn clauses (Buntine, 1986a, 1986b) presented in the second section.

We then propose a new framework for the induction of concepts when logic-based description languages are to be used. Basically, we view induction as a process of model-directed discovery of consistent patterns in the data observed from the standpoint of our relevant background knowledge. By noting patterns in the form of constraints and rules, such induction would gradually converge to a single hypothesis (or more precisely - a class of equivalent hypotheses). This framework allows for instance a much cleaner formulation of the factoring technique (Subramanian and Feigenbaum, 1986).

In the fourth section we present a new algorithm for constructing generalisations that addresses the problem of discovering rules from the data. The plausible generalisation algorithm, based on a method of constructing most specific generalisations in the presence of arbitrary Horn clause background knowledge, accounts for the need for constraining irrelevance when generalisations are made utilizing the background knowledge. The algorithm can be easily tailored for instance to perform a search of a class of models applicable to a particular sequence prediction induction problem. The importance of *relevance* to this algorithm is highlighted and the basic techniques for determining relevance reviewed.

# 1..1. Horn Clauses as a Language for Induction

We believe that Horn clause logic provides an ideal framework for the investigation and practice of logic-based induction. A good deal of previous work can be rephrased into the language of Horn clauses. For example, Bundy et al. (1985) compare a number of very different induction techniques by describing them in this unifying context. Similarly, Mooney (1986) uses Horn clauses when analysing a related learning technique, Explanation-Based Learning.

Both *disjunctive* and *conjunctive* concepts can be described in the language. Background knowledge provides the means for *additional descriptors* to be derived about any example. *Recursion* is an important feature in any language requiring to solve sequence prediction problems and is also essential to conveniently express concepts about recursively defined structures such as trees, lists and language grammars. The language also has a well understood means of *performing inferences* (Lloyd, 1984), essential to derive relevant descriptors from background knowledge during the induction process. This is critical to the development of a computational model of generality.

The techniques for incorporating *negation* into this language have not been considered here (see for example Lloyd, 1984). Negation should be a fundamental component of any general concept description language. One method of induction that allows negation has been suggested by Vere (1980). However, to the author's current knowledge, a complete analysis of the effects of negation on the induction process has not yet been made.

# 2. A MODEL OF GENERALITY

To perform logic-based induction, certain tools are required. Induction is usually considered a process of searching for descriptions consistent with the background knowledge and known examples (Michalski, 1983). The search space for descriptions can be partially ordered by the generality of possible descriptions (Mitchell, 1982; Dietterich et al., 1982). So to practice induction of Horn clauses, we need tools that can manipulate the generality of Horn clauses and move around in the resultant search space.

Only recently has a thorough analysis been made of this search space for Horn clauses based on a model of generality incorporating arbitrary Horn clause background knowledge (Buntine, 1986a, 1986b). This was inspired by the work of (Sammut and Banerji, 1986). In this section, we introduce this model; it is used throughout the paper. Previous models of generality have considered the case where (1) background knowledge is in the form of facts (Plotkin, 1971; Vere, 1977); (2) in the form of equalities on predicates, type hierarchies and some additional forms (Kodratoff and Ganascia, 1986); and (3) in the form of concept hierarchies (Mitchell et al., 1981). Background knowledge of an equivalent kind to that considered here has been employed in conceptual clustering (Michalski and Stepp, 1986). Michalski (1983) stresses the importance of background knowledge for practical induction and mentions a number of generalisation and specialisation rules.

Before describing the notion of generality considered here, we introduce the terms used.

## 2.1. Preliminaries

Corresponding to a pure Prolog rule, a *Horn clause*, usually abbreviated to clause or called a *rule*, is in the form

$$\forall_{\mathbf{X}}(\mathbf{A}_{0}\leftarrow\mathbf{A}_{1}\wedge\ldots\mathbf{A}_{n})$$

where  $n\geq 0$ , the A<sub>i</sub>'s are *atoms* (*predicate symbols* applied to some arguments, see the examples below) and X is the set of variables occurring in the A<sub>i</sub>'s. The universal quantification is usually implicit. The atoms on the right hand side of a rule are referred to as the *conditions* for the rule.

#### For n=0, a clause is called a *fact* and written simply as $A_0$ .

The *description* of a concept takes the form of a *predicate definition*, a set of clauses with the same predicate symbol on the left hand side of the rule. This predicate symbol is used to give examples of the concept. For instance, the concept "friends", denoting who is friends with whom, could be specified by the following definition for the two place predicate friends.

friends(bob,mary) friends(mary,bill) friends(X,Y)←friends(Y,X)

Informally, the task of induction is to construct a predicate definition such that all and only positive examples of the concept will be proven true by a series of successful applications of the clauses in the predicate definition and in the current Horn clause background knowledge.

A second kind of knowledge construct used here is a *constraint*. Constraints are introduced because they are a useful kind of knowledge about concepts that can feasibly be induced from data and, as we shall see, they can also be used to simplify the induction of rules. The model of generality in its current state is not able to account for constraints computationally however.

In this paper, we consider constraints of the form

$$\forall_{\mathbf{X}}(\mathbf{A}_{0} \rightarrow \exists_{\mathbf{Y}_{1}}(\mathbf{A}_{1,1} \land \dots \land \mathbf{A}_{1,n1}) \lor \dots \exists_{\mathbf{Y}_{m}}(\mathbf{A}_{m,1} \land \dots \land \mathbf{A}_{m,nm})) \quad \text{or} \quad \forall_{\mathbf{X}}(\sim \mathbf{A}_{0})$$

where:  $A_0$  and the  $A_{i,j}$ 's are atoms; X is the set of variables occurring in  $A_0$ ; the number of conjuncts in the disjunction on the right hand side of the constraint,  $m, \ge 1$ ; and for the i-th conjunct,  $Y_i$  is the set of remaining variables and  $n_i \ge 1$ . Again, the quantification is usually implicit. In the database literature these would be referred to as *tuple generating dependencies* (Sagiv, 1986) when m=1. These constraints could, for instance, be checked at each step of a computation.

Each constraint is actually the *converse* of a particular set of rules. To see this, swap the direction of the arrow in the constraint form given above then distribute " $\vee$ " over " $\leftarrow$ ". For instance, consider the following predicate definition which may be suggested to explain instances of the sort predicate. N.B. sort(X,Y) is true if the list Y is a sorted version of the non-empty list X.

 $sort(X,Y) \leftarrow X = Y = [U]$ 

 $sort(X,Y) \leftarrow Y = [A,B|L] \land A \le B$ 

Giving "sort" its usual interpretation, this description of sort is incorrect as the second rule says sort([3,4],[1,2]) is true. The description is logically equivalent to

 $sort(X,Y) \leftarrow (X=Y=[U]) \lor (Y=[A,B|L] \land A \leq B)$ 

and taking the converse we get the constraint

 $\operatorname{sort}(X,Y) \rightarrow \exists_U(X=Y=[U]) \lor \exists_{A,B,L}(Y=[A,B|L] \land A \leq B)$ 

which is a useful constraint on the sort predicate.

# 2.2. The Model

We adapt Mitchell's view of "more general" (1982) as follows. For a fuller treatment of the model refer to (Buntine, 1986b).

Informally, a first description is *more general* than a second if it is possible to explain, using the current background knowledge, that the first description will apply to some example whenever the second description does. We give three different but equivalent views of this notion below.

A predicate definition R1 is *more general* than another R2 with respect to given background knowledge, if, in any possible world (sometimes refered to formally as an *extension* or *interpretation*) consistent with the background knowledge, whenever a clause from R2 can be successfully applied to show, in the possible world, an example of the concept should be a positive example, then a clause from R1 must be able to be successfully applied to show the same. We say a clause C1 is more general than another clause C2 when the equivalent condition for clauses occurs.

Alternatively, if R1 is represented in the form  $\forall_X(c(X) \leftarrow B1(X))$  and R2 as  $\forall_X(c(X) \leftarrow B2(X))$ were "c" is the concept being described and B1 and B2 represent the bodies of the descriptions (in the case of predicate definitions, a disjunction of conjunctions), then R1 is more general than R2 with respect to the background knowledge if and only if

 $\forall_{\mathbf{X}} (B2(\mathbf{X}) \rightarrow B1(\mathbf{X}))$ 

is a logical consequence of the background knowledge.

If a first clause or predicate definition is more general than a second, then the second is also said to be *more specific* than the first.

To illustrate these views of generality and later ideas, we have concocted the following background knowledge, labelled "the social knowledge-base".

likes(bill,mary) likes(bob,jane) right\_handed(bill) right\_handed(bob) friends(bill, paul) friends(kate, bob) friends(paul, sally) friends(paul, mary) friends(jane, kate) friends(sally, kate) friends(X,Y)←friends(Y,X) plays\_tennis(mary) member\_tennis\_social\_club(jane) member\_tennis\_social\_club(X)←plays\_tennis(X) plays\_social\_tennis(X)←member\_tennis\_social\_club(X)

Under this background knowledge, the description {likes(bill,mary), likes(bob,jane)} is more specific than the description consisting of the single rule

 $likes(X,Y) \leftarrow member\_tennis\_social\_club(Y) \land friends(X,Z) \land friends(Z,Y)$ (1)

To justify this, we must show that rule 1 confirms likes(bill,mary) and likes(bob,jane) in any possible world consistent with the "social knowledge-base". That is, we must show that for (X,Y) set to (bill,mary) or (bob,jane), the right hand side of rule 1 will be true in any possible world consistent with the social knowledge base. In logic, this is equivalent to showing that rule 1 together with the "social knowledge-base" logically implies likes(bill,mary) and likes(bob,jane).

An operational view of generality follows. In this view, background knowledge can only consist of Horn clauses.

It has been shown (Buntine, 1986b) that, with respect to given Horn clause background knowledge, a predicate definition R1 is more general than another, R2, if and only if for every clause in R2, there exists a clause in R1 more general than it. That is, to test the comparative generality of two predicate definitions, we only need compare the generality of their component rules.

Furthermore, a test of the comparative generality of two clauses consists of a single query to the logic program representing the background knowledge (Buntine, 1986b). Effectively, a rule is more general than another with respect to given Horn clause background knowledge if the more general rule can be converted to the other by repeatedly turning variables to constants, adding conditions or partially evaluating by applying some rule in the background knowledge to the R.H.S. When the background knowledge is absent, the test degenerates to a test of  $\theta$  subsumption. (Plotkin, 1970). Clause C  $\theta$ -subsumes D if there exists a set of replacements for the variables in C (a substitution) making the atoms in C a subset of those in D. That is, clause C can be converted to clause D by turning some variables to constants and adding conditions.

This model of generality then formalises and considerably extends the work of Plotkin (1970,1971), Vere (1977), and to some extent Kodratoff and Ganascia (1986). We denote this model of generality *generalised subsumption*.

# 3. APPLICATIONS

## 3.1 Knowledge Maintenance

Redundancy detection is one of a number of knowledge maintenance tasks. It has been argued (Kitakami et al., 1984) that knowledge maintenance should be a task performed by an integrated knowledge acquisition system. Recent experiences with XCON indicate (Pundit, 1985) that knowledge maintenance will become a key task in the future.

In (Buntine, 1986b) it is shown that the model of generality given previously has applications to the detection of redundancy in rule bases. We are currently developing techniques for detecting *unused* rules and *redundant rules*, for removing *redundant conditions* from a rule's right hand side and for generally cleaning-up the form of a rule. These techniques are based on a stronger model of generality than generalised subsumption, a model incorporating background knowledge in the form of constraints as well as rules. Related results derived using a different technique, based on the notion of *equivalence* rather than *generality*, appear in the deductive database literature (Sagiv, 1986).

What does the model have to say about the search space for induction?

## 3.2. Induction

A number of techniques have been devised by the AI community for searching a space of descriptions ordered on generality. Some of these techniques allow a description to consist of only a single conjunctive rule, others allow disjunction.

Some of the techniques are based on a specific to general search, for instance climbing the generalisation hierarchy (Sammut and Banerji, 1986), constructing maximally specific generalisations of individual pairs of rules in the current description (Vere, 1977) and hybrid techniques (Vrain, 1986; Kodratoff and Ganascia, 1986).

The second technique above is based on the notion of a *most specific generalisation* (MSG). For instance, the MSG of two clauses C1 and C2 with respect to background knowledge P is a clause that is a generalisation of both C1 and C2 with respect to P but is more specific than any other such generalisation. In the context of a number of different representations, MSGs have been employed in various ways as a useful induction tool (Vere 1977, 1980; Mitchell et al., 1981; Fu and Buchanan, 1985).

Other search techniques are based on a general to specific search (Langley, 1981; Shapiro, 1981, 1983). The process of specialising a rule found to be inconsistent with new negative examples, performed during general to specific search, is called *discrimination* (Langley, 1981; Bundy et al., 1985). A number of different search styles have been suggested. Mitchell (1982), Langley (1981) and Shapiro (1981) use a breadth first search, Michalski (1983) a beam search and Bundy et al. (1985) suggest a depth first search.

As specific to general search has the effect of constructing plausible rules suggested by the data whereas general to specific search constrains potential rules from being overly general, it is now well accepted that these complementary search strategies be coupled together. The two most widely known techniques are *Version Spaces* (Mitchell, 1982) and *Focusing* (Bundy et al., 1985). A recent improvement to the Version Spaces algorithm, justifying that the space can be split into independent components, is the technique of *factoring* (Subramanian and Feigenbaum, 1986). This splitting is already found in the Focusing algorithm.

A further kind of search is *model-directed* search, discussed and applied to the induction problem by (Dietterich et al., 1986).

In the following subsections, we consider the two search strategies general to specific and specific to general in the light of the generalised subsumption model and practical experience. For the discussion, it is sufficient to consider Version Spaces and Focusing in their decoupled form.

**Performing Specific to General Search.** We first consider the problem of performing a specific to general search of the space of predicate definitions.

Given any background knowledge expressed as Horn clauses, there exists a method of climbing the corresponding generalisation hierarchy (Buntine, 1986a). The Marvin system (Sammut and Banerji, 1986) demonstrates a practical induction method based on a similar technique.

However, many of the techniques mentioned in the previous subsection use a potentially more powerful method of performing specific to general search - constructing MSGs. This allows much larger jumps up the generalisation hierarchy to be made, jumps that are well justified given that the search space contains a correct hypothesis. This is done for instance in the specific to general component of the Version Spaces algorithm.

In its usual sense, the MSG of two predicate definitions is, simply, the union of the sets of clauses they are constructed from. For example, the MSG of the two descriptions

 $\{\operatorname{member}(1,[1])\}$ 

 $\{member(2, [2, 3]), member(2, [3, 2])\}$ 

is

 $\{member(1,[1]), member(2,[2,3]), member(2,[3,2])\}.$ 

As this achieves no induction gain, we can instead form a generalisation by additionally constructing the MSG of some pair of clauses from the predicate definitions. For instance, the previous two could be generalised to

{member(X, X.Y), member(2,[3,2])}

The first atom in this set is the MSG of member(1,[1]) and member(2,[2,3]) with respect to no background knowledge.

This is the idea behind the control strategy of Vere's Thoth (1977). In this way, MSGs of clauses can be usefully employed when performing specific to general search of the space of predicate definitions. In effect, we concurrently perform several specific to general searches of the space of rules.

Recently, an algorithm for constructing the shortest MSG of two clauses, in the presence of arbitrary Horn clause background knowledge, has been developed (Buntine, 1986c). This was based on some ideas presented in (Buntine 1986a). Experiences with the algorithm brought to light various problems which we show below.

One problem occurring is that an MSG does not always exist, in which case it can be thought of as existing but being an infinite clause. This happens in quite common situations.

For example, an MSG of the example facts member(4,[3,4]) and member(2,[5,1,2]) with respect to the background knowledge

list([]) list(X.Y)←list(Y) member(X,X.Y)←list(Y) member(4,[3,4]) member(2,[1,2]) member(3,[5,3]) member(2,[5,1,2])

is actually an infinite clause. The trouble here is that an infinite number of facts can be inferred from the background knowledge which are able to be associated directly or indirectly with the two examples. For instance, the following clauses are some successive partial results of the MSG algorithm mentioned.

member(X,Y.Z.W). member(X,Y.Z.W)←list(W)∧member(X,Z.W)∧member(X,[A.X]). member(X,Y.Z.W)←list(W)∧member(X,Z.W)∧member(X,[A.X]) ∧member(2,[B,2])∧member(B,1.W)∧member(C,[1,B]) ∧member(X,[D,B])∧member(C,[1,2])∧member(X,[D,E]) + 8 more terms

A variation on this theme is that an abundance of *irrelevant* knowledge (using the term in a similar sense to Michalski, 1983) can be inferable from the background knowledge. An MSG of the known facts likes(bill,mary) and likes(bob,jane) with respect to the "social knowledge base" is:

likes(A,B)←friends(A,C)∧friends(C,B)∧friends(C,sally) ∧member\_tennis\_social\_club(B)∧right\_handed(A)

Right-handedness is not usually a property causing one person to be attracted to another. Commonsense tells us it is irrelevant in this situation. So this rule could be made more plausible by removing the condition about right-handedness. When a larger less selective knowledge base is used the small amount of irrelevance in the above clause could become considerably greater.

To get around these problems and still retain the power of the MSG approach, we can modify the algorithm to constrain some of the possible conditions that could be included on the right hand side of the constructed clause. The aim is to make generalisations which are as specific as can be consistent with an acceptably terse solution and information about relevance. This results in the *Plausible Generalisation Algorithm* (PGA), discussed in some detail in the fourth section.

The *control* of any such constructive generalisation algorithm approximating MSGs requires extra care however. Because MSGs do not always exists, this complication is necessary.

When performing specific to general search of a space of rules using an algorithm approximating MSGs, the exact control regimes used by methods such as Version Spaces or Focusing in their specific to general component will not be appropriate. The use of such an algorithm will require backtracking when the approximations made are more general than needed. When using an algorithm approximating MSGs to generalise descriptions consisting of several rules, in a similar manner to Vere's Thoth mentioned previously, additional control will again be required.

Specific to general search in the presence of background knowledge has a feature already mentioned however that further complicates this problem of control - the problem of *constraining irrelevance*. That is, of the multitude of facts that can be derived from the

background knowledge about given examples, which should not be used when constructing generalisations? And, how do we identify them?

This involves providing a facility to constrain the derived facts that can be used when constructing generalisations. As it happens, this also provides a convenient mechanism on which to build a flexible model-directed search for rules. The facility is discussed in the fourth section. Experience with PGA indicates that its use is *essential* for practical induction when a body of sometimes irrelevant background knowledge has been supplied.

Though we have left a number of problems unsolved, the following can be said with certainty. To generalise a description, a set of rules, we can approximate the MSG of a pair of rules in the description. Sometimes the MSG can be constructed exactly, othertimes we need to constrain its construction using notions of relevance and/or of size. Unfortunately, this confounds the problem of control. Further related issues will be considered in the fourth section.

**Performing General to Specific Search.** We next consider the problem of general to specific search of the description space.

Firstly, we will argue that, unlike some logic-based methods used, a *heuristic* and/or *model-directed* general to specific search is required in practice. Evidence is presented below, based on the work of Shapiro. Utgoff's approach of revising the description language (1986) is founded on a related premise - the enormity of the inductive search space.

Experiences with Shapiro's Model Inference System indicate that a breadth first search, in the case of Horn clauses, is highly impractical. Shapiro was actually working with a model of generality incorporating no background knowledge (called subsumption, mentioned in section 3). But in this case of general to specific search it has sufficiently similar characteristics to the stronger model used here. The *generalisation hierarchy*, the general to specific search space, still has a huge branching factor. A relevant point is that the search spaces considered by Bundy et al., Langley, and Mitchell et al. (1981) are smaller by comparison due to the restricted languages/generality models used.

Shapiro effectively employed a model-directed technique to attempt to overcome the problem of a huge branching factor. For instance, many of the examples given by him used a model where no constants appear in the conditions of clauses (see appendix II, PDSREF, in Shapiro, 1983). Instantiating constants is of course one of several rule specialisation steps so this had the effect of reducing the generalisation hierarchies branching factor considerably. Another model he used is definite clause grammars (DCGREF).

This model-directed technique is useful in the situation where we have a prior belief that the description should belong to a particular class of hypotheses (for a related discussion, see Dietterich et al., 1986).

Another technique possible is the heuristic search of the general to specific space. It is at this point where the *credibility of hypotheses* plays an important role (Rendell, 1986). A key factor for success in Quinlan's ID3 (Quinlan, 1986b) and its more recent version C4 (Quinlan, 1986a) is their techniques for determining how to expand the decision tree. They tend to, but not always, develop a shorter tree, sometimes trading off accuracy for a decrease in size (Quinlan, 1986a). There are strong arguments to suggest that without any other source of prior information, this produces a more creditable hypothesis (Georgeff and Wallace, 1985; Cheeseman, 1984). Likewise, heuristic techniques for the more extensive search problem considered here need to be developed.

These two techniques, model-directed and heuristic are not at odds with each other. We believe they should work together in collaboration.

Secondly, we note that a general to specific search of the space of rules can also be thought of as a specific to general search of the space of constraints. This alternative view allows a much cleaner formulation of the search problem.

This change in formulation is allowed because of the following property. We say a predicate definition is *overly general* if it can be used to prove true all possible positive examples of the concept, and in addition some negative ones as well. All members of the G-set for the Version

Spaces algorithm for instance are overly general. If a predicate definition for a concept is overly general, then its *converse* actually represents a constraint that must be true of the concept. A proof of this is quite straight forward.

Therefore, the technique of coupled specific to general and general to specific search can also be thought of as a specific to general search of the space of rules together with a specific to general search of the space of constraints. Constraints balance rules and vice versa, as in the coupled paradigm. Induction then, becomes a process of searching for and generalising consistent patterns in the data, regardless of whether those patterns are expressed as rules or constraints.

One advantage of phrasing the search in terms of constraints is that the the question "What is the strength of your belief that constraint C applies to concept S?" is more natural to consider than the question "What is the strength of your belief that clause C is overly general for concept S?" Heuristic search of the general to specific search space of descriptions essentially involves repeatedly asking this question. The mathematician Polya, whose work was a prime inspiration behind the AM program (Lenat, 1983), demonstrates that reasonable heuristics may exist on a search for constraints (Polya, 1954). He gives a number of examples, with suitable heuristics, in the domain of number theory.

Another advantage is that search of the space of constraints can be made more efficient as search can proceed *independently* for different constraints. Any number of constraints induced can later be combined to produce a single more powerful constraint. This result rests on the following property: Given two valid constraints about concept "c"

$$\forall_{\mathbf{X}}(\mathbf{c}(\mathbf{X}) \rightarrow \exists_{\mathbf{Y}_{1}}(\mathbf{A}_{1,1} \land \dots \land \mathbf{A}_{1,ni}) \lor \dots \exists_{\mathbf{Y}_{m}}(\mathbf{A}_{m,1} \land \dots \land \mathbf{A}_{m,nm}))$$

 $\forall_{\mathbf{X}}(\mathbf{c}(\mathbf{X}) \rightarrow \exists_{\mathbf{Y}_{1}}(\mathbf{B}_{1,1} \land \dots \mathbf{B}_{1,n1}) \lor \dots \exists_{\mathbf{Y}_{p}}(\mathbf{B}_{p,1} \land \dots \mathbf{B}_{p,np}))$ 

then the following is also a valid constraint

$$\forall_{\mathbf{X}} \Big( c(\mathbf{X}) \rightarrow \Big( (\exists_{\mathbf{Y}_{1}} (\mathbf{A}_{1,1} \land \dots \land \mathbf{A}_{1,n1}) \lor \dots \exists_{\mathbf{Y}_{m}} (\mathbf{A}_{m,1} \land \dots \land \mathbf{A}_{m,nm}) \Big) \\ \land \Big( \exists_{\mathbf{Y}_{1}} (\mathbf{B}_{1,1} \land \dots & \mathbf{B}_{1,n1}) \lor \dots \exists_{\mathbf{Y}_{p}} (\mathbf{B}_{p,1} \land \dots & \mathbf{B}_{p,np}) \Big) \Big) \Big)$$

but stronger than the previous two. Considering that the converse of these constraints represent sets of rules about concept "c", this means that the general to specific search can be decomposed into independent components. In the case where neither constraint has a disjunction, the resultant technique mimics *factoring*, an improvement to the Version Spaces algorithm proposed independently by Subramanian and Feigenbaum (1986).

In summary, we suggest that the notion of a general to specific search of the description space be also considered as a specific to general search of the space of constraints. This can potentially improve search efficiency as many constraints can be searched for independently. Whether search is of the space of descriptions or constraints, it needs to be model-directed and heuristic - guided by a prior beliefs about hypotheses and their current credibility in view of known examples.

# 4. THE PLAUSIBLE GENERALISATION ALGORITHM

Having discussed the induction problem in general, we now consider a particular sub-problem: generalising plausible rules from examples.

This section introduces and discusses the *Plausible Generalisation Algorithm* (PGA). Using this algorithm, the useful generalisation

$$ember(X, Y, Z, W) \leftarrow member(X, Z, W) \land list(W)$$

for the member example introduced in section 3.2 can be quickly constructed using a notion of relevance such as allowing only smaller lists to appear in the R.H.S. of the rule.

The algorithm is intended to be used in the following way (similar to Vere's Thoth). We initialise the description of the concept to the set of positive examples. We repeatedly apply PGA to pairs of rules in the description to obtain a more general description, backtracking when over generalisation occurs. We do not consider, in this paper, how this process would be controlled or when it would be terminated.

Before introducing the algorithm, an important practical issue must first be covered. An MSG by generalised subsumption is, strictly, never unique. However, all possible MSGs are equivalent to one another under the partial order "generalised subsumption". A kind of canonical form (called *reduced* - after Plotkin, 1970, see Buntine 1986b) exists for these possibilities that represents the shortest possible clause. A longer version of the most specific generalisation of likes(bill,mary) and likes(bob,jane) given previously is

likes(A,B)  $\leftarrow$  friends(A,C)  $\land$  friends(C,B)  $\land$  friends(C,sally)

 $\land$ member\_tennis\_social\_club(B) $\land$ right\_handed(A)

 $\beta$  plays\_social\_tennis(B) $\beta$ friends(B,C) $\beta$ friends(C,D)

The condition plays\_social\_tennis(B) would be a consequence of the condition member\_tennis\_social\_club(B) so is a *redundant condition* in the clause. Likewise for the two conditions following it. Fortunately, these conditions can often be detected during the generalisation process (in step 2c of the algorithm in figure 1, see also Buntine, 1986c).

# 4.1 An Informal Introduction

Below, we describe the algorithm informally, applied to the common situation where the clauses being generalised are merely positive examples of the concept. We consider the situation of finding a plausible generalisation of the positive examples likes(bill,mary) and likes(bob,jane) with respect to the social knowledge base.

We will have to consider the following kinds of arguments. These have a certain psychological appeal about them suggesting they may be useful for a broader class of induction problems coping with background knowledge.

To find some common reason for bill liking mary and bob liking jane, we must find potentially relevant statements common about both bill and bob and/or mary and jane. To do this we need to query the knowledge base to find out for instance "what do bill and bob have in common?"

Of course not all answers to this query will be relevant to the situation that bill likes mary and bob likes jane so we will need some *relevance constraints* to prune obviously irrelevant details. Such a constraint would prevent such details as bill and bob's right-handedness from being considered. For instance, we may only allow facts concerning social-life of people to be considered relevant here (the actual mechanics of checking relevance is in step 2b of the algorithm in figure 1)

Furthermore, some answers will be *a joint consequence* of existing facts we have already found and so be redundant. We notice that mary and jane are both members of the tennis social club and so add this to the conditions on the right hand side of the rule. But because it is a consequence of this that they also both play social tennis, we don't need to consider their playing social tennis in the constructed rule.

Finally, as we are concerned with constraining the size of the clause constructed, certain controls are provided. A bound is provided on the depth that inference can go when searching for relevant statements (mentioned in step 2a of the algorithm in figure 1). Another bound is provided on the depth of nesting of existential quantification (mentioned in step 2e).

# 4.2 The Algorithm

The actual algorithm proceeds as follows.

We first construct the left hand side of the final generalisation (step 1 of the algorithm). This is done by finding the most specific atom which has likes(bill,mary) and likes(bob,jane) as instances of it. This is called the atom *generalising* the two facts. We construct it using Plotkin's algorithm to find the least generalisation of terms (Theorem 1, 1970). In this case it is the atom likes(A,B) where the variables A,B could take the values bill and mary or bob and jane. The ordered pair (bill,bob) is called the *conflict pair* for A and (mary,jane) likewise for B.

Our task of induction is to find potentially relevant statements common to both bill and bob and/or mary and jane, that is, common to corresponding members of a conflict pair. So far, our rule must look as follows.

## likes(A,B) $\leftarrow$ {some conditions on A and B}

The conditions on the right hand side must currently be known to hold true when A and B take the values bill and mary or bob and jane, so that the rule can be shown to be a generalisation of the initial examples.

From the background knowledge it can be seen that

member\_tennis\_social\_club(mary), member\_tennis\_social\_club(jane)

and friends(bill, paul), friends(bob, kate)

are potentially relevant statements. Before using these, we must check to see if they are relevant (step 2b) and that they wont be redundant in the final clause (step 2c).

We construct the atoms generalising the pairs of facts found and link them to the rule to produce

 $likes(A,B) \leftarrow member_tennis_social_club(B) \land friends(A,C)$ 

 $\wedge$  {further conditions on A, B and now possibly C}

Notice that when generalising the atom we create a new conflict pair (paul,kate) associated with the variable C. Common statements about paul and kate could indirectly effect bill and bob so these must also be considered.

Proceeding in this fashion, we gradually build up potentially relevant conditions on the left hand side of the rule, discarding irrelevant or redundant conditions as we go. The final rule would look as follows:

 $likes(A,B) \leftarrow friends(A,C) \land friends(C,B) \land member_tennis_social_club(B)$ 

We summarise the algorithm in figure 1. A more exact account of the algorithm and the theory behind it occurs in (Buntine, 1986c).

1. Construct the left hand side and find the initial set of conflict pairs.

e.g. likes(A,B) with conflict pairs (bill,bob) and (mary,jane) for variables A and B respectively.

2. Construct the right hand side:

Repeat

2a. Look for a pair of facts inferable from the background knowledge (searching to some depth n) possibly relevant to corresponding members of a conflict pair.

e.g. the pair friends(bill,paul) and friends(bob,kate) are possibly relevant to bill and bob.

e.g. the pair friends(bill,paul) and friends(kate,bob) are not as bill and bob do not occur in corresponding positions.

2b. Disregard the pair (so back to 2a) if they are considered irrelevant by the relevance constraints.

2c. Disregard the pair (so back to 2a) if they have been derived using exactly the same rule (this is a sufficient condition for them to be redundant in the final rule constructed).

2d. Construct the atom generalising the pair of facts.

e.g. friends(A,C).

2e. Add the atom to the conditions accumulated so far. Add any new conflict pairs found to the set of conflict pairs if they are not too indirect.

e.g. (paul,kate) is a new conflict pair for C.

Until step 2a fails to find a pair of facts that hasn't already been considered.3. Strip away any redundant conditions from the right hand side of the constructed rule.

Fig. 1 - The Plausible Generalisation Algorithm

# 4.3 Features of the Algorithm

The algorithm has several noteworthy features we outline below.

**Constraining Irrelevance.** It is our experience with the algorithm, in a number of small scale experiments involving induction of simple list concepts (union etc.) and simple sequence prediction problems, that the control of *irrelevant* knowledge by providing *relevance* constraints is vital for the algorithms effective use. Support for the use of relevance constraints comes from successful experiences reported with ID3 (Quinlan, 1984), its descendent C4 (Quinlan, 1986a) and various commercial offspring (Michie, 1986; Hart, 1985). Using the ID3 paradigm, it has been found that experts usually have the ability to articulate knowledge in the form of tutorial examples and relevant attributes (usually, features about examples that are true or false). If their same ability scales up to the induction problem as it is phrased in Horn clause logic, then experts should also be able to provide knowledge about relevance of the kind suggested here.

Some useful relevance constraints we have found handy for inducing list related concepts say to only use facts about: lists of smaller length; sub-lists; or lists containing only constants occurring in the given example. Relevance constraints for sequence prediction problems can for example specify that the current member in the sequence is dependent on only the previous L elements (termed *lookback* by Dietterich et al., 1986), or is periodic with some specified period.

Thus, relevance constraints are also a means of specifying a particular model in which to perform induction. In our system, written in Prolog, a new model can be specified simply by redefining the *relevance* predicate. This provides great flexibility for performing model-directed search using classes of predefined and indeed user defined models.

**Commonality Queries.** Critical to the algorithm's performance are queries to the knowledge-base of the form "What statements are known common to both (objects) X and Y?" occurring in step 2a. In our system we currently use a brute force approach. We refer to these as *commonality queries*. Knowledge-base systems can feasibly be tuned to answer this basic style of query.

Of course, potential answers can be further constrained by the need for relevance. So one could also ask: "What statements are known common to both (objects) X and Y that are potentially relevant to the statements (or areas of concern) S and T?" This is asking "what is of interest to the current induction problem?" We believe efficient answering of such queries will be an important pre-requisite for practical inductive inference systems sourcing knowledge from a knowledge-base.

**Determining Irrelevance.** Central to the use of relevance constraints or the asking of commonality queries is the ability to determine what facts are actually relevant. Clearly, relevance is related to the notion of statistical dependence. Relevance is also strongly tied to the notion of causality. If all events in the world have some cause, then the occurrence of one event can only be relevant to the occurrence of another if they share, directly or indirectly, a common cause.

**Sufficiency of the Knowledge.** A feature of generalisations found by PGA is that one consistent with the intended concept can only be constructed if sufficient knowledge is available (can be derived in step 2a of the algorithm) from which the generalisation can be justified.

This can be a problem with recursively defined concepts. For example, the MSG of member(4,[3,4]) and member(2,[5,1,2]) with respect to the background knowedge

list([]). list(X.Y)←list(Y). member(X,X.Y)←list(Y). member(4,[3,4]). member(2,[5,1,2]).

is member(X,Y.Z.W)  $\leftarrow$  list(W) which would be inconsistent with most negative examples as it says anything is a member of a list with more than 1 element. The problem here is that the fact member(2,[1,2]) is not available so the valid generalisation

member(X,Y.Z.W)  $\leftarrow$  member(X,Z.W)  $\land$  list(W). can not be justified from the existing knowledge.

This kind of problem occurs whenever the concepts on which a predicate definition is to be based are themselves not fully known. Of course, this can be used to advantage i.e. to constrain irrelevance.

Operation of PGA then must proceed under the *justifiability assumption* (Buntine 1986b). That is, it is assumed that the instructor has supplied sufficient knowledge from which valid generalisations of at least some seed facts can be justified and hence constructed.

# 4.3. Comparison with other Generalisation Frameworks

A method of constructing common generalisations also using background knowledge is presented by Vrain (1986). The notion of relevance constraints provides an alternative to Utgoff's approach for specifying bias (1986).

As an aside, it is interesting to compare this technique with Explanation-Based Learning (DeJong, 1986; Mitchell et al., 1985). Explanation-based Learning proceeds in a knowledge rich environment where the generalisation steps can be plausibly if not fully justified. This occurs in situations where the system has more knowledge of what is being learnt than just positive and negative examples (as is the case here). Background knowledge is being used by PGA to implicitly suggest explanations. Though these explanations cannot be justified by PGA, they can be kept from being implausible by careful use of relevance constraints.

# 5. CONCLUSION

We have based the paper on a theory of generality for Horn clauses where generality is set in the context of background knowledge in the form of facts and Horn clauses.

Guided by this, we have shown that some existing induction techniques are inadequate for the induction of descriptions in the extended context where background knowledge is available. This is due to a number of reasons. In the presence of background knowledge, most specific generalisations, often calculated by these methods, may be infinite or weighted down with irrelevant conditions. Furthermore the full Horn clause search spaces, for the general to specific search particularly, are huge and search needs to be model-directed and/or heuristic. These problems can only be magnified when more extensive knowledge representations are used.

Though we haven't presented a completed replacement technique we have rephrased and redirected the existing techniques. We view induction as the model-directed discovery of patterns, i.e. constraints and rules, in the data. The problem of heuristic search of the space of constraints can be factored into independent searches for constraints and necessarily raises the kinds of questions "How strongly do you believe constraint C is valid?" The problem of heuristic search of these inductive search spaces remains an open problem.

We have also presented an algorithm, the Plausible Generalisation Algorithm that is intended to form the basis of a technique for discovering rules in the data. The algorithm has been implemented in Prolog and experiments performed on small scale induction problems illustrate the basic problems involved with this approach.

The algorithm generalises examples to form rules. The algorithm accepts additional input in the form of relevance constraints that allow a trainer to focus the algorithm on relevant information rather than all possible information inferable from the knowledge base. Success in simpler induction frameworks indicates that relevance is one kind of knowledge an expert can articulate more readily than actual rules. These constraints also allow simple and flexible specification of a model in which a model-directed induction could proceed.

Finally, we have identified problems that must be considered if induction is to be performed drawing on a body of background knowledge i.e. constructive induction. One is the problem of determining irrelevance. The second is the answering of queries of the form "What do (objects) X and Y have in common that is relevant to situation Z?"

## Acknowledgements

Thanks to Ross Quinlan and Prof. Donald Michie for their suggestions about an earlier draft of this paper and Jenny Edwards, John Hughes and Graham Wrightson for their support.

## References

- Bundy, A., Silver, B., Plummer, D., (1985), "An Analytical Comparison of Some Rule Learning Programs", Artificial Intelligence, 27.
- Buntine, W.L., (1986a), "Towards a Practical Theory of Horn Clause Induction", Proc. of 9th Annual Australian Computer Science Conf., Canberra.
- Buntine, W.L., (1986b), "Generalised Subsumption and its Applications to Induction and Redundancy", to appear Artificial Intelligence. This is an extended version of a paper with the same title in *Proc. European Conference on Artificial Intelligence*, (1986), Brighton, U.K.
- Buntine, W.L., (1986c), "A Most Specific Generalisation Algorithm for Horn Clauses", draft in preparation.
- Cheeseman, P., (1984), "Learning of Expert Systems from Data", Proc. IEEE Workshop on Principles of Knowledge-Based Systems, Denver.
- Cheeseman, P., (1986), "Induction of Models Under Uncertainty", unpublished manuscript.
- DeJong, G., (1986), "A Brief Overview of Explanatory Schema Acquitision", In: Mitchell, T.M., Carbonell, J., Michalski, R., (Eds.), *Machine Learning - A Guide to Current Research*, Kluwer Academic.
- Dietterich, T.G., London, R., Clarkson, K. and Dromey, R., (1982), "Learning and Inductive Inference". In: Cohen, P. and Feigenbaum E. (Eds.), *The Handbook of Artificial Intelligence*, Morgan Kaufmann, Los Altos.
- Dietterich, T.G., Michalski, R.S., (1986), "Discovering Patterns in Sequences of Objects", In: Michalski, R.S., Carbonell, J., and Mitchell, T.M., (Eds.) Machine Learning - An Artificial Intelligence Approach, Vol. II, Morgan Kaufmann, Los Altos.
- Fu, L.-M. and Buchanan, B.G., (1985), "Learning Intermediate Concepts in Constructing a Hierarchical Knowledge Base". Proc. 9th Int. Joint. Conf. Artificial Intelligence, UCLA.
- Georgeff, M.P., Wallace, C.S. (1985), "A General Selection Criterion for Inductive Inference", SRI Technical Note 372.
- Hart, A., (1985), "The Role of Induction in Knowledge Elicitation". Expert Systems, 2.
- Kitakami, H., Kunifuji, S., Miyachi, T. and Furukawa K., (1984), "A Methodology for Implementation of a Knowledge Acquisition System". Proc. IEEE Int. Symp. on Logic Programming, Atlanta City.
- Kodratoff, Y., Ganascia J-G., (1986), "Learning as a Non-deterministic but Exact Logical Process", In: Michalski, R.S., Carbonell, J., and Mitchell, T.M., (Eds.) *Machine Learning* An Artificial Intelligence Approach, Vol. II, Morgan Kaufmann, Los Altos.
- Langley, P., (1981), "Language Acquisition through error recovery", CIP working paper 43, Carnegie-Mellon Uni.
- Lenat, D.B., Davis, R., (1982), Knowledge-Based Systems for Artificial Intelligence, McGraw-Hill.

6-13

Lloyd, J.W., (1984), Foundations of Logic Programming, Springer-Verlag.

Michalski, R., (1983), "A Theory and Methodology of Inductive Learning", Artificial Intelligence, 20.

Michalski, R., Stepp, R.E., (1986), "How to Structure Structured Objects", In: Michalski, R.S., Carbonell, J., and Mitchell, T.M., (Eds.) *Machine Learning - An Artificial Intelligence Approach, Vol. II*, Morgan Kaufmann, Los Altos.

- Michie, D., (1986), "Current Developments in Expert Systems". 2nd Australian Conference on Applications of Expert Systems, Sydney.
- Mitchell, T.M., (1982), "Generalisation as Search". Artificial Intelligence, 18, pp203-226.
- Mitchell, T.M., Mahadevan, S., Steinberg, L.I., (1985), "LEAP: A learning Apprentice for VLSI Design". Proc. 9th Int. Joint. Conf. Artificial Intelligence, Los Angeles.
- Mitchell, T.M., Utgoff, P.E., Nudel, B., Banerji, R., (1981), "Learning Problem Solving Heuristics through Practice". Proc. 7th Int. Joint. Conf. Artificial Intelligence, Vancouver.
- Mooney, R., (1986), "A Domain Independent Explanation-based Generaliser", AAAI-86, Proc. 5th National Conf. on Artificial Intelligence, Philadelphia.
- Plotkin, G.D., (1970), "A Note on Inductive Generalisation". In: Michie, D., (Ed) Machine Intelligence 5, pp153-163, Elsevbier North-Holland, New York.
- Plotkin, G.D., (1971), "A Further Note on Inductive Generalisation". In: Michie, D., (Ed) Machine Intelligence 6, pp101-124, Elsevbier North-Holland, New York.
- Pundit, N., (1985), "On AI Applications", Invited Talk, Conf. on Commercial Applications of Expert Systems, (Sydney).
- Quinlan, J.R., (1984), "Learning Efficient Classification Procedures and their Application to Chess End games". In: Michalski, R.S., Carbonell, J., and Mitchell, T.M., (Eds.) Machine Learning An Artificial Intelligence Approach, Springer-Verlag.

Quinlan, J.R., (1986a), "Inductive Knowledge Acquisition: A Case Study", 2nd Australian Conference on Applications of Expert Systems, Sydney.

Quinlan, J.R., (1986b), "Induction of Decision Trees", Machine Learning, 1(1).

.

- Rendell, L., (1986), "A General Framework for Induction and a Study of Selective Induction", *Machine Learning*, 1(2).
- Robinson, J.A., (1979), Logic Form and Function The Mechanisation of Deductive Reasoning, North-Holland, New York.
- Sagiv, Y., (1986), "Optimizing Datalog Programs". In: Minker J. (Ed.), Workshop on Foundations of Deductive Databases and Logic Programming, Washington.
- Sammut, C.A. and Banerji, R.B., (1986), "Learning Concepts by Asking Questions". In: Michalski, R.S., Carbonell, J., and Mitchell, T.M., (Eds.) Machine Learning - An Artificial Intelligence Approach, Vol. II, Morgan Kaufmann, Los Altos.
- Shapiro, E.Y., (1981), "Inductive Inference of Theories from Facts", TR 192, DCS, Yale University.
- Shapiro, E.Y., (1983), Algorithmic Program Debugging, MIT Press.
- Subramanian, D., Feigenbaum, J., (1986), "Factorization in Experiment Generation", AAAI-86, Proc. 5th National Conf.on Artificial Intelligence, Philadelphia.
- Utgoff, P., (1986), "Shift of Bias for inductive Concept Learning", In: Michalski, R.S., Carbonell, J., and Mitchell, T.M., (Eds.) *Machine Learning - An Artificial Intelligence Approach, Vol. II*, Morgan Kaufmann, Los Altos.
- Vere, S.A., (1977), "Induction of Relational Productions in the Presence of Background Information", *Proc. 5th Int. Joint. Conf. Artificial Intelligence*, Cambridge MA.
- Vere, S.A., (1980), "Multilevel Counterfactuals for Generalisations of Relational Concepts and Productions", *Artificial Intelligence*, 14, pp139-164.
- Vrain, C., (1986), "The use of Domain Properties Expressed as Theorems in Machine Learning", Proc. Int. Meeting on Advances in Learning, Les Arcs, France.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# Generic Tasks for Knowledge-Based Reasoning: The "Right" Level of Abstraction for Knowledge Acquisition\*

Tom Bylander and B. Chandrasekaran Laboratory for Artificial Intelligence Research Department of Computer and Information Science The Ohio State University Columbus, Ohio, USA 43210

#### Abstract

Our research strategy has been to identify generic tasks -- basic combinations of knowledge structures and inference strategies that are powerful for dealing for certain kinds of tasks. Our strategy is best understood by considering the "interaction problem," that the representation of knowledge strongly interacts with the inference strategy that is applied to the knowledge and with the task that the knowledge is used for. The interaction problem implies that different knowledge acquisition methodologies will be required for different kinds of reasoning, e.g., a different knowledge acquisition methodology for each generic task. We illustrate this using the generic task of hierarchical classification. Our proposal and the interaction problem call into question many generally held beliefs about expert systems, such as the belief that the knowledge base should be separated from the inference engine.

## Introduction

Knowledge acquisition is the process that extracts knowledge from a source (e.g. a domain expert or textbook) and incorporates it into a knowledge-based system. Because a knowledge-base representation is its target, knowledge acquisition cannot be separated from a broader theory of knowledge-based reasoning. A solution to knowledge acquisition must be compatible with a solution to the general problem of knowledge-based reasoning.

For some time now, we have been developing a *theory of generic tasks* that identifies several types of reasoning that knowledge-based systems perform and provides a overall framework for the design and implementation of such

<sup>&</sup>lt;sup>\*</sup>Research supported by Air Force Office of Scientific Research, grant 82-0255, National Science Foundation grant MCS-8305032, and Defense Advanced Research Projects Agency, RADC Contract F30602-85-C-0010.

systems [Chandrasekaran 83, Chandrasekaran 84, Chandra 86 86]. In this paper, we present our theory as a way to exploit the interaction problem. Because each generic task exploits it differently, each one should be be associated with a different knowledge acquisition methodology.

First, we pose and discuss the "interaction problem." Next, we review our theory of generic tasks: the characteristics of a generic task and the generic tasks that have been identified so far. In view of the interaction problem, we propose our theory of generic tasks as a framework for identifying different knowledge acquisition methodologies. We illustrate this using the generic task of hierarchical classification. Finally, we reflect on a number of beliefs that have driven much of the past research on knowledge acquisition and knowledge-based reasoning.

### The Interaction Problem

The interaction problem is this:

The representation of knowledge strongly interacts with the inference strategy that is applied to the knowledge and with the task that is performed with it.

In other words, knowledge representations have a close relationship to how they are used to solve problems. Knowledge is dependent on its use. The interaction problem is not a new concept. Minsky, in his famous frame proposal, argues that "factual and procedural contents must be more intimately connected to explain the apparent power and speed of mental activities" [Minsky 75 - p. 211]. Marr has noted that "how information is represented can greatly affect how easy it is to do different things with it" [Marr 82 - p. 21]. Our argument takes a different perspective, that the inference strategy and task influence what knowledge that is represented, i.e., we will represent knowledge so that it takes advantage of how we are going to use it.

The interaction problem, if true, has serious implications for how knowledge acquisition should be done. Because some knowledge representation must be the target of knowledge acquisition, knowledge acquisition methodologies must take the interaction problem into account. Also, if different kinds of reasoning have different kinds of interactions, there is a need for a different knowledge acquisition methodology for each kind of reasoning.

#### The Interaction Problem for General Knowledge Representations

Each of the major forms of knowledge representation -- rules, logic, and frames -are subject to the interaction problem. Knowledge is affected by the task (different tasks require different knowledge) and by inference strategy (knowledge is adapted to the strategy). In addition to giving examples of the interaction problem, we also wish to emphasize that the generality of a representation does not make the interaction problem disappear.

The Interaction Problem in Rules. The idea of rules is to explicitly map situations to actions. Naturally then, the focus is on determining what conditions characterize the situations and what conclusions characterizes the actions. The result is that two different tasks in the same domain can have different rules representing the "same" knowledge. For example in diagnosis, rules of the form "symptom --> malfunction" will be implemented, while in prediction of symptoms, the rules will be in the form "malfunction --> symptom". In each case, the rules will be tuned to the task that is being performed. One might argue that there is no problem with keeping both tasks in mind, and so both kinds of rules can be implemented at the same time. Of course, given that one has already taken the interaction problem into account, the knowledge base then will have rules appropriate for the tasks to be performed.

Another source of interaction is that special programming techniques are needed to encode task-specific inference strategies. For example, R1 [McDermott 82], which is implemented in OPS5 [Forgy 81], performs a sequence of design subtasks, each of which is implemented as a set of production rules. However, OPS5 has no construct equivalent to a subtask, so the grouping of rules and the sequencing from one set of rules to another are achieved by programming techniques. Clearly, R1's task structure has had an significant effect on how knowledge was encoded in OPS5's production rule representation.

Different inference strategies for rules are also a source of interaction. If EMYCIN's backward-chaining strategy is used, rules can combine with other rules to increase or decrease confidence in a given conclusion [van Melle 79]. On the other hand, if OPS5's recognize-and-act strategy is used, only one rule at a time can fired, so that situations must be matched to actions much more exactly. Also, the "context" must be carefully controlled to ensure that appropriate rules are considered. Note that the difference is not whether EMYCIN does forward- or backward-chaining, but that EMYCIN allows rules to act in parallel, while OPS5 applies rules in serial.

The Interaction Problem in Logic. Rule-based and logic-based representations are fairly similar with respect to the interaction problem. Like rules, logic provides for a direct way for drawing conclusions from situations. In the context of a specific task, it is only useful to encode propositions that can make task-relevant conclusions. Propositions for a diagnostic system would be like "if symptom A, then maybe malfunctions X or Y or Z," while a prediction system would have propositions like "if malfunction X, then maybe symptoms A or B or C." This example also shows one danger in applying both kinds of propositions indiscriminately. Given some confidence in malfunction X, then some confidence in symptom A should be inferred, followed by inferring some confidence in malfunctions Y and Z, which probably wasn't intended. Logic-based representations are also like rules with respect to implementing the structure of a task and dealing with different inference strategies. To implement R1 in predicate logic, for example, a subtask construct would also have to be implicitly programmed. Two different inference strategies for logic, such as PROLOG and resolution theorem proving, are quite different to use.

The Interaction Problem in Frames. The emphasis in frame representations is on describing the conceptual structure of the domain. However, different tasks might need quite different conceptual structures. For example, classificatory problem solving [Gomez and Chandrasekaran 81, Clancey 85] in general needs a generalization hierarchy (hypothesis-subhypothesis), while routine design [Brown and Chandrasekaran 84] in general needs a structural hierarchy (component-subcomponent).

Frames are intended to flexibly interact with the inference strategy. After all, the idea of procedural attachment is to embed procedures in the frames so that the appropriate inferences are triggered.

#### Exploiting the Interaction Problem

The interaction problem will not go away no matter what representation is chosen. Every knowledge-based system will be developed, debugged, and maintained so its knowledge works with its inference strategy and so its knowledge in combination with its inference strategy solves a certain set of problems. No one undertakes an exhaustive study of a domain, i.e., acquires any and all the knowledge associated with that domain. It would take too long and we humans are too lazy anyway. We have learned that only a selected portion of domain knowledge needs to be acquired to perform specific tasks.

Instead of trying to lessen the impact of the interaction problem, our research strategy has been to *exploit* it. Our strategy is not new; exploiting the interaction problem has been the "untold story" of knowledge-based systems, and perhaps of AI in general. This should be obvious to anybody who has ever maintained a knowledge-based system. To find and correct an error, one has to understand both the problem solving and the knowledge base, and how they combined to cause the error.

It should not be surprising that different representations can be exploited in different ways and are thus more applicable to certain kinds of tasks than others. This is where our theory of generic tasks comes in. Our intent is to propose kinds of reasoning in which the representation and the inference strategy can be exploited to solve certain kinds of tasks. For a particular domain and task, our intent is to encode a selected portion of domain knowledge into an efficient and maintainable problem solving structure.

#### The Proposal

Intuitively one would think that diagnosis in different domains would have certain types of reasoning in common, and that design in different domains would also have certain types of reasoning in common, but that diagnostic reasoning and design problem solving will be generally speaking different. For example, diagnostic reasoning generally involves malfunction hierarchies, rule-out strategies, setting up a differential, etc., while design involves device/component hierarchies, design plans, ordering of subtasks, etc. However, the formalisms (or equivalently the languages) that have been commonly used for knowledge-based systems do not capture these distinctions. Ideally, diagnostic knowledge should be represented by using the vocabulary that is appropriate for diagnosis, while design knowledge should have a vocabulary appropriate for design. Our approach to this problem has been to identify generic tasks -- basic combinations of knowledge structures and inference strategies that are powerful for dealing for certain kinds of tasks. The generic tasks provide a vocabulary for describing reasoning tasks, as well as for designing knowledge-based systems that perform them.

Characterization of a Generic Task

Each generic tasks is characterized by information about the following:

- 1. The type of *task* (the type of input and the type of output). What is the function of the generic task? What is the generic task good for?
- 2. The *representation* of knowledge. How should knowledge be organized and structured to accomplish the function of the generic task?
- 3. The *inference strategy* (process, problem solving, control regime). What inference strategy can be applied to the knowledge to accomplish the function of the generic task?
- 4. The type of *concepts* that are involved in the generic task. What concepts are the input and output about? How is knowledge organized in terms of concepts? How does the inference strategy operate on concepts? In essence, we adopt Minsky's idea of frames as a way to organize the problem solving process [Minsky 75].

The phrase "generic task" is somewhat misleading. What we really mean is an elementary generic combination of a task, representation, and inference strategy about concepts. The power of this proposal is that if a problem matches the function of a generic task, then the generic task provides a knowledge representation and an inference strategy that can be used to solve the problem. It should be noted that a problem might match the function of more than one generic task, so that several strategies might be used to solve the problem, depending on the knowledge that is available. Also, generic tasks can be composed for more

complex reasoning, i.e., a generic task may call upon another generic task to solve a subproblem.

#### Examples of Generic Tasks

Our group has identified several generic tasks. Here, we briefly describe the generic tasks of hierarchical classification [Gomez and Chandrasekaran 81] and object synthesis by plan selection and refinement [Brown and Chandrasekaran 84].

*Hierarchical Classification.* Task: Given a description of a situation, determine what categories or hypotheses apply to the situation.

Representation: The hypotheses are organized as a classification hierarchy in which the children of a node represent subhypotheses of the parent. There must be knowledge for calculating the degree of certainty of each hypothesis.

Inference Strategy: The establish-refine strategy specifies that when a hypothesis is confirmed or likely (the establish part), its subhypothesis should be considered (the refine part). Additional knowledge may specify how refinement is performed, e.g. to consider common hypotheses before rarer ones. If a hypothesis is rejected or ruled-out, then its subhypotheses are also ruled-out.

Important Concepts: Hypotheses.

Examples: Diagnosis can often be done by classification. In planning, it is often useful to classify a situation as a certain type, which then might suggest an appropriate plan. MYCIN [Shortliffe 76] can be thought of as classifying a patient description into an infectious agent hierarchy. PROSPECTOR [Duda *et al.* 80] can be viewed as classifying a geological description into a type of formation.

Object Synthesis by Plan Selection and Refinement. Task: Design an object satisfying specifications. An object can be an abstract device, e.g. a plan or program.

Representation: The object is represented by a component hierarchy in which the children of a node represent components of the parent. For each node, there are plans that can be used to set parameters of the component and to specify additional constraints to be satisfied. There is additional knowledge for selecting the most appropriate plan and to recover from failed constraints.

Inference Strategy: To design an object, *plan selection and refinement* selects an appropriate plan, which, in turn, requires the design of subobjects at specified points in time. When a failure occurs, failure handling knowledge is applied to make appropriate changes.
Important Concepts: The object and its components.

Examples: Routine design of devices and the synthesis of everyday plans can be performed using this generic task. The MOLGEN work of Friedland [Friedland 79] can be viewed in this way. Also R1's subtasks [McDermott 82] can be understood as design plans.

# Other Generic Tasks

Other generic tasks that have been identified include knowledge-directed information retrieval [Mittal et al. 84], abductive assembly of explanatory hypotheses [Josephson et al. 84], hypothesis matching [Chandrasekaran et al. 82], and state abstraction [Chandrasekaran 83]. More detail on the overall framework can be found in [Chandra 86 86].

## Exploiting Classificatory Problem Solving

Each generic task exploits domain knowledge in a different way; it calls for knowledge in a specific form that can be applied in a specific way. Because the knowledge acquisition methodology must be able to extract and select the appropriate knowledge, each generic tasks calls for a different knowledge acquisition methodology. For illustration we consider the generic task of hierarchical classification. In classification, the emphasis is on obtaining the classification hierarchy that contains the hypotheses that are relevant to the task and adaptable to the strategy. This section does not provide a complete knowledge acquisition methodology for classification, but outlines a number of considerations that a methodology must take into account. Additional guidelines for using classification can be found elsewhere [Mittal 80, Bylander and Smith 85].

# Determining Hypotheses of Interest

Classificatory problem solving is useful for determining the hypotheses that apply to a situation. The first step then is to decide upon the hypotheses that the problem solver should potentially output. For example in diagnosis, the potential malfunctions of the object should be considered. The goal here is to determine the specific categories that should be produced, so if a general category is considered (e.g. "something is wrong with X"), then more specific categories should be generated (e.g. by asking "What types of problems can occur with X?"). Determining the usefulness of a category is discussed below.

## Analyzing Commonalities among Hypotheses

Once a collection of classificatory hypotheses have been identified, one needs to determine the commonalities among the hypotheses. These commonalities become potential candidates for mid-hierarchy hypotheses in the classification hierarchy.

The easiest example to handle is when one hypothesis is clearly a subhypothesis of another, i.e., it asserts a more specific category. In general, two hypothesis may have commonalities along the following lines:

- Definitional The two hypotheses share a definitional attribute, e.g., hepatitis and cirrosis are liver diseases. Rain and snow are forms of precipitation.
- Appearance The two hypotheses are recognized using common pieces of evidence. Both cholestasis and hemolytic anemia have jaundice as a common symptom. Wet grass is symptomatic of both rain and dew.
- Planning The two hypotheses are associated with similar plans of action. Both the common cold and allergies are reasons to take plenty of facial tissue with you. Either lightning or strong winds are good reasons for staying inside.

The ideal hypothesis asserts some definitional attribute over all its subhypotheses, has an appearance common to all its subhypotheses, and also provides constraints on the plans associated with its subhypotheses.

In general, the hierarchy should follow a definitional decomposition whenever possible. However, there are cases where appearance is an important consideration. For example, the Dubin-Johnson syndrome is a benign hereditary disorder that mimics key symptoms of cholestasis (jaundice, conjugated hyperbilirubinemia - high amounts of conjugated bilirubin in the blood). Because it *looks* so much like cholestasis, it is most useful to make it a subhypothesis of cholestasis.

## Assessing Evidence for or against Hypotheses

The above two steps should generate a large number of hypotheses. However, not all of them will be useful for classificatory problem solving, i.e., there is a need to select a classification hierarchy that can be used to exploit the establish-refine strategy, getting rid of any intermediate hypothesis do not provide additional problem solving power. Because the language we have used for classification, CSRL, requires a classification *tree* [Bylander and Mittal 86], we have become familiar with some of the strategies for evaluating hypotheses. However, the following questions are relevant whether a tree or tangled hierarchy is used.

- Are there sufficient criteria to distinguish the hypothesis from other hypotheses? In other words, does this hypothesis have a different appearance from other hypotheses?
- Is there evidence that distinguishes the hypotheses from its siblings? Because the establish-refine strategy does not consider a hypotheses unless its parent (or one of its parents in a tangled hierarchy) is

relevant, evidence that distinguishes the hypothesis from its siblings is especially important.

• Is the evidence normally available? Evidence for or against an hypothesis is not very useful if is not likely to be available to the system when it is running. For example in medical diagnosis, some tests are relatively risky, expensive, or time-consuming to perform, so it is best to use hypotheses that rely on outward signs and symptoms and generally available laboratory data.

We have generally used another generic task, hypothesis matching, for mapping evidence to confidence values in hypotheses [Chandrasekaran *et al.* 82]. However, we do not want to complicate the central issue by considering combinations of generic tasks. Examples of how hypothesis matching can be exploited are provided in [Sticklen *et al.* 85] and [Bylander and Mittal 86].

## Debugging Hypotheses

An important part of knowledge acquisition is being able to find out what knowledge was incorrect or left out when something goes wrong. In classification, the following problem can occur.

- Wrong confidence value Debug the knowledge that produces the confidence value. Sticken *et al.* [Sticklen *et al.* 85] describes how hypothesis matching can be debugged. The problems below assume that the confidence values are reasonable in view of the evidence considered.
- Bad hierarchical structure If a hypothesis was incorrectly considered or incorrectly left unconsidered, the problem may be simply that the hypothesis is in the wrong place in the hierarchy. The hypothesis is not, by definition or appearance, a subhypothesis of its parent (or some other ancestor).
- Failure to consider Another reason why a hypothesis might not be considered is because an ancestor was not refined. The problem might be that there isn't enough evidence to support a high level of confidence in the ancestor. In this case, a restructuring is necessary in order to provide a way for establish-refine to reach the hypothesis. Less drastic solutions are lowering the threshold for refining the ancestor or considering more evidence for the ancestor.
- Establish-refine strategy is too simple Sometimes a hypothesis should not be considered even if its parent is established. For example, if one of the hypothesis's siblings is confirmed, and the hypothesis is incompatible with its siblings, then the hypothesis should not be considered. The solution here is to adapt the establish-refine strategy to

take this additional information into account. It should be noted that this problem is not a defect of establish-refine. Instead, it shows that establish-refine is really a *family* of strategies. The CSRL language, for example, provides a default establish-refine strategy and allows other establish-refine strategies to be defined.

# The Point of Classification

The real reason for doing hierarchical classification is to obtain the hypotheses that describe a situation. Therefore, we should adopt knowledge acquisition methodologies that are intended to produce efficient and maintainable classification systems. To do this, we need to exploit the interactions between the representation, inference strategy, and the task. For classification, it means that each potential hypotheses of a classification hierarchy must be evaluated with respect to how well it interacts with the establish-refine strategy and whether the task may need to output it.

# A Reexamination of Past Beliefs

Some generally held beliefs about knowledge-based systems need to be reexamined in light of the interaction problem and our proposal to exploit it. These beliefs have served the first generation of knowledge-based systems well, especially in stimulating much research and discussion. However, we believe it is the time to reconsider them.

## Belief #1: Knowledge should be uniformly represented and controlled.

This belief denies the interaction problem and implies that there is nothing to be gained by using different representations to solve different problems. Our experience is that when the problems of a domain match the generic tasks, the generic tasks provide explicit and powerful structures for understanding and organizing domain knowledge.

## Belief #2: The knowledge base should be separated from the inference engine.

This belief denies that the inference strategy affects how knowledge is represented. However, its real effect has been to force implementors to implicitly encode inference strategies within the knowledge base. Both MYCIN, whose diagnostic portion is best understood as classification, and R1, which is best understood as routine design, show that this separation is artificial.

# Belief #3: Control knowledge should be encoded as metarules.

Although metarules address the problem of how to have multiple, explicit strategies in a rule-based system, the metarule approach ignores other aspects of the interaction problem. The "separation of control knowledge from domain knowledge" promotes the view that domain knowledge can be represented independent of its use, i.e., that different sets of metarules can be applied as needed. However, given a clear strategy (whether metarules or inference engine) and a task to be performed, the domain knowledge will be adapted to interact with the strategy to solve the task.

# Belief #4: The ontology of a domain should be studied before considering how to process it.

We believe that ontology should not be performed just for its own sake, but in view of the tasks that need to be done. For example, to apply classification to a domain, there is a need to focus on the hypothesis space and evaluate hypotheses. Although other knowledge structures (e.g. component hierarchies, causal networks) may be useful for other generic tasks, if classification is going to be performed, then knowledge acquisition should concentrate on those aspects of the domain that are relevant to classification. This is not to say that a domain shouldn't be analyzed to identify what generic tasks are appropriate; however, this kind of domain analysis does not require an exhaustive ontology of the domain.

## Belief #5: Correct reasoning is a critical goal for knowledge-based systems.

Everything else being equal, being correct is better than being incorrect. However, an emphasis on correctness detracts from more critical issues. One of those issues is developing an understanding of the appropriate strategies to be applied to a task. For example, there has been much research and debate about normative methods for calculating uncertainty. The reasoning problem, though, is not how to precisely calculate uncertainty, but how to avoid doing so. In diagnosis, e.g., there is much more to be gained by using abduction (assembling composite hypotheses to account for symptoms), then by independently calculating the degree of certainty of each hypothesis to several decimal places of accuracy.

# Belief #6: Completeness of inference is a critical goal for knowledge-based systems.

Everything else being equal, being complete is better than being incomplete, but an emphasis on completeness ignores the fact that certain kinds of inferences will be more important than others for a particular task. For example in our description of classification, we did not mention that when a subhypothesis is confirmed, one can infer that its ancestors are also confirmed. However, that inference is not the crucial aspect of classification. The important inference is that when a hypothesis is confirmed or likely, then its subhypotheses should be considered.

# Belief #7: A representation that combines rules, logic, frames, etc. is what is needed.

Such representations appear to be a good compromise since they let you represent knowledge in the "paradigm" of your choice. Unfortunately, this is, at best, only

an interim solution until something better is found. None of the individual representations fully address the interaction problem, nor do they distinguish between different types of reasoning.

## Generic Tasks at the "Right" Level of Abstraction

The first generation of research into knowledge-based systems has conducted a extensive search for a "holy grail" of representation, in which knowledge could be represented free of assumptions of how it would be used. For any particular task, though, certain kinds of inferences will become critical to the task, and consequently, domain knowledge needs to be organized so those inferences are performed efficiently. This is how the interaction problem arises, and why it will never go away. Instead of futilely trying to avoid it, the interaction problem needs to be studied and understood so that methods of exploiting it can be discovered and applied.

Our theory of generic tasks is an attempt to provide the "right" level of abstraction for this and other problems of knowledge-based reasoning. Each generic task provides a knowledge structure in which knowledge can be organized at a conceptual level. In classification, the concepts are hypotheses organized as a classification hierarchy. Each generic task identifies a combination of a task definition, representation, and inference strategy that exploits the interaction problem. We have shown how the generic task of classification can be associated with a knowledge acquisition methodology that takes advantage of the interactions between domain knowledge and classificatory problem solving.

## References

[Brown and Chandrasekaran 84]

D. C. Brown and B. Chandrasekaran.

 Expert Systems for a Class of Mechanical Design Activity.
 In Proc. IFIP WG5.2 Working Conference on Knowledge Engineering in Computer Aided Design. IEEE Computer Society, Budapest, Hungary, 1984.

[Bylander and Mittal 86]

T. Bylander and S. Mittal.

CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling.

AI Magazine 7(2):66-77, 1986.

[Bylander and Smith 85]

T. Bylander and J. W. Smith.

Mapping Medical Knowledge into Conceptual Structures.

In Proc. Expert System in Government Symposium, pages 503-511. IEEE Computer Society, McLean, Virginia, 1985.

[Chandra 86 86]	<ul> <li>B. Chandrasekaran.</li> <li>Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design.</li> <li><i>IEEE Expert</i> 1(3):23-30, 1986.</li> </ul>
[Chandrasekaran	B. 83] B. Chandrasekaran. Towards a Taxonomy of Problem Solving Types. AI Magazine 4(1):9-17, 1983.
[Chandrasekaran	84] B. Chandrasekaran. Expert Systems: Matching Techniques to Tasks. Artificial Intelligence Applications for Business. Ablex, Norwood, New Jersey, 1984, pages 116-132.
[Chandrasekaran	<ul> <li>et al. 82]</li> <li>B. Chandrasekaran, S. Mittal, and J. W. Smith.</li> <li>Reasoning with Uncertain Knowledge: The MDX Approach.</li> <li>In Proc. Congress of American Medical Informatics Association, pages 335-339. AMIA, San Francisco, 1982.</li> </ul>
[Clancey 85]	W. J. Clancey. Heuristic Classification. Artificial Intelligence 27(3):289-350, 1985.
[Duda et al. 80]	<ul> <li>R. O. Duda, J. G. Gaschnig, and P. E. Hart.</li> <li>Model Design in the Prospector Consultant System for Mineral Exploration.</li> <li>Expert Systems in the Microelectronic Age.</li> <li>Edinburgh University Press, 1980, pages 153-167.</li> </ul>
[Forgy 81]	C. L. Forgy. OPS5 Users Manual. Technical Report CMU-CS-81-135, Carnegie-Mellon University, 1981.
[Friedland 79]	<ul> <li>P. Friedland.</li> <li>Knowledge-based Experiment Design in Molecular Genetics.</li> <li>PhD thesis, Computer Science Department, Stanford University, 1979.</li> </ul>
[Gomez and Ch	andrasekaran 81] F. Gomez and B. Chandrasekaran. Knowledge Organization and Distribution for Medical Diagnosis. <i>IEEE Trans. Systems, Man and Cybernetics</i> SMC-11(1):34-42, 1981.

[Josephson et al. 84]

J. R. Josephson, B. Chandrasekaran, and J. W. Smith. Assembling the Best Explanation. In Proc. IEEE Workshop on Principles of Knowledge-Based Systems,

- pages 185-190. IEEE Computer Society, Denver, 1984.
- [Marr 82] D. Marr. Vision. W. H. Freeman, 1982.

[McDermott 82] J. McDermott.

R1: A Rule-based Configurer of Computer Systems. Artificial Intelligence 19(1):39-88, 1982.

[Minsky 75] M. Minsky.

A Framework for Representing Knowledge. The Psychology of Computer Vision. McGraw-Hill, 1975, pages 211-277.

[Mittal 80]

S. Mittal.
Design of a Distributed Medical Diagnosis and Database System.
PhD thesis, Dept. of Comp. and Info. Sci., The Ohio State University, 1980.

[Mittal et al. 84]

 S. Mittal, B. Chandrasekaran, and J. Sticklen.
 Patrec: A Knowledge-Directed Database for a Diagnostic Expert System.
 Computer 17(9):51-58, 1984.

[Shortliffe 76] E. H. Shortliffe.

Computer-Based Medical Consultations: MYCIN. Elsevier, New York, 1976.

[Sticklen et al. 85]

 J. Sticklen, B. Chandrasekaran, and J. W. Smith.
 MDX-MYCIN: The MDX Paradigm Applied to the MYCIN Domain.
 Computers and Mathematics with Applications 11(5):527-539, 1985.

[van Melle 79] W. van Melle.

- A Domain Independent Production-Rule System for Consultation Programs.
- In Proc. Sixth Interational Conf. on Artificial Intelligence, pages 923-925. Tokyo, 1979.

# Acquisition of Uncertain Rules in a Probabilistic Logic

John G. Cleary Dept. Computer Science, University of Calgary, 2500 University Dr., Alberta T2N 1N4, Canada.

uucp: cleary@calgary ARPA:cleary.calgary.ubc@csnet-relay CRNET: cleary@calgary.cdn CSNET:cleary.calgary@ubc

## ABSTRACT

The problem of acquiring uncertain rules from examples is considered. The uncertain rules are expressed using a simple probabilistic logic which obeys all the axioms of propositional logic. By using three truth values (true, false, undefined) a consistent expression of contradictory evidence is obtained. As well the logic is able to express the correlations between rules and to deal both with uncertain rules and with uncertain evidence. It is shown that there is a subclass of such rules where the probabilities of correlations between the rules can be directly computed from examples.

## UNCERTAIN RULES

Uncertainty is an important part of many ruled based expert systems. For example, applications such as medical diagnosis do not allow anything but the weakest inferences to be made from available evidence. To report certain conclusions is both incorrect and misleading. A number of schemes for expressing and computing such uncertainties have been developed. For example, in MYCIN and EMYCIN "certainty factors" are used (Shortliffe, 1976). A certainty factor is a number between -1 and +1, -1 is intended to express sure knowledge that something is false and +1 sure knowledge that it is true. Intermediate values express varying degrees of ambivalence about the truth. For example 0 expresses a complete lack of knowledge about truth or falsity. Such certainty factors can be used in a number of ways. For example a rule such as:

will-rain  $\leftarrow$  dark-cloud and falling-pressure with certainty 0.6;

says that it is almost certainly true that if there is dark cloud around and the barometric pressure is falling then it will rain, although, there will be some cases where this is not true. Certainty factors can be propagated through the system to evaluate the certainty of conclusions. For example the conclusion 'will-rain' would be given a certainty factor based on the certainty of the rule above and the certainties of 'dark-cloud' and 'falling-pressure'. Unfortunately there are grave problems with using certainty factors in this way. Consider the additional rule:

will-rain  $\Leftarrow$  lightning with certainty 0.4;

If both these rules fire then 'will-rain' is given a higher certainty factor than if only one of them fires. Unfortunately the second rule is merely another way of saying that storm clouds are present and the conclusion is not much more true as a result.

These problems can be seen more starkly by considering the expression '(dark-cloud or not dark-cloud)' or the expression '(dark-cloud and not dark-cloud)' which are respectively always true or false. However certainty factors ignore the fact that the two parts of the expression are correlated (the same) and report intermediate values for both expressions.

Another way to approach this is to use the probability that an expression is true rather than certainty factors. Again problems arise that are similar to those above. For example, one scheme used to compose such probabilities is obtained by assuming the various parts of an expression are uncorrelated:

$$p \text{ and } q = p \times q$$
  

$$p \text{ or } q = p + q - p \times q$$
  

$$not p = 1-p$$

Let p=1/2 then (p and not p) = 1/4 and (p or not p) = 3/4, neither of which is correct.

A second evaluation scheme (Zadeh, 1965) assumes that some correlation can occur between expressions and lets:

p and q = min(p,q)p or q = max(p,q)not p = 1-p

Again let p=1/2 then (p and not p) = 1/2 and (p or not p) = 1/2 which are both wrong. This scheme does not overweight rules which are similar but does underweight rules which are independent.

Another weaker scheme (Quinlan, 1983) uses intervals of probabilities. It is based on the principle that the following inequalities always hold:

 $max(0,1-p-q) \le (p \text{ and } q) \le max(p,q)$  $max(p,q) \le (p \text{ or } q) \le min(1,p+q)$  $not \ p=1-p$ 

By only asserting that the probability for an expression lies in some range it never asserts anything which is false. For example if p = 1/2 it deduces that:

 $0 \le (p \text{ and } not p) \le 1/2$ 

and  $1/2 \le (p \text{ or not } p) \le 1$ .

While true these inequalities are too weak to be generally useful.

A related problem is that some conclusions may have evidence both indicating that they are true and indicating that they are false. It seems, intuitively, that the situation where there is no evidence about something is different from the one where there is a known 0.5 probability that it is true and a known 0.5 probability that it is false. The schemes mentioned above also have problems in consistently accomodating both positive and negative evidence.

Considerations such as these show that it does not seem to be possible to provide a quantitative theory of truth and falsity using just probability values for expressions. In the next section an alternative is introduced which circumvents this by using (potentially infinite) sequences of bits to represent the truth or falsity of a statement. Probabilities can

be extracted after reasoning is complete but the calculations cannot take place using just the probabilites. Other attempts to provide a logical basis for uncertain reasoning have been made (Shapiro, 1983), (van Emden, 1986) but because they are based on the probability values of expressions they also are heir to the ills described above.

# A PROBABILISTIC LOGIC

The technique used here is to assign each expression an infinite sequence of true/false values rather than just one true/false possibility. This is trivially different from the normal propositional calculus. All theorems hold, for example, (p or not p) = (true, true,...) and (p and not p) = (false,false, ...). In order to make this useful a new family of logical constants are introduced. Each constant is some infinite sequence of true/false values with a fixed probability that it will be true. The constants are written in the form  $\tau(x)$  where x is the probability that an item in the sequence is true. So, it is always true that  $\tau(1) = (true, true, ...)$ 

When evaluating a logical expression with these constants some actual value has to be assigned to them. There are a number of ways of doing this and it is convenient to assume that different constants are uncorrelated. That is, the probability of a true in the sequence  $(\tau_1(x) \text{ and } \tau_2(y))$  is always x×y. In practice these infinite uncorrelated sequences are likely to be approximated by finite sequences of pseudo-randomly generated bits.

To express an uncertain rule the constants can be used as follows:

will-rain  $\Leftarrow$  dark-cloud and falling-pressure and  $\tau(0.6)$ ;

This says that if there is dark cloud and falling barometric pressure then in 0.6 of the cases there will be rain. The second rule can be expressed as:

will-rain  $\Leftarrow$  lightning and  $\tau(0.4)$ ;

This has still not solved the problem that the two rules are highly correlated but they can be reformulated as:

$$\begin{pmatrix} \text{will-rain} \leftarrow \text{dark-cloud and falling-pressure and } \tau_1(0.75) \\ \text{and} \\ \text{will-rain} \leftarrow \text{lightning and } \tau_2(0.5) \end{pmatrix} \leftarrow \tau_3(0.8)$$

This reformulation says that the two rules have a common cause which says that they are true 80% of the time. As a result of this the probability of will-rain will only be weakly augmented when both rules fire, solving the original problem of expressing the fact that the two rules are not independent of each other. By rewriting these rules in the equivalent form below it can be seen that the original  $\tau(0.6)$  has been replaced by ( $\tau_1(0.75)$  and  $\tau_3(0.8)$ ) and  $\tau(0.4)$  by ( $\tau_2(0.5)$  and  $\tau_3(0.8)$ ):

will-rain  $\Leftarrow$  dark-cloud and falling-pressure and  $\tau_1(0.75)$  and  $\tau_3(0.8)$ will-rain  $\Leftarrow$  lightning and  $\tau_2(0.5)$  and  $\tau_3(0.8)$ 

In this way sets of rules with arbitrary correlations between them can be expressed. However, it is not clear that all possible ways of using the logical constants are useful. The form used above where sets of rules are enabled seems to be an intuitive way of expressing such realtionships and, as will be seen later, it has some advantages when the probabilities of the constants are acquired from experience. However, the acid test for such formalisms

is whether computer naive experts can express their intuitions in this form. No tests of this have been done nor has there been exploration of possible ways of "sugaring" the syntax to make it more palatable.

Probabilities can be extracted from our logical sequences by counting the number of true values in the sequence and taking the limit. The result of all this is a probability logic (Gaines, 1984), (Rescher, 1963). The logic obeys all the usual logical axioms including the tautology (x or not x).

## Rule Sets

As is usual in rule based systems, it is necessary in practice to restrict attention to the Horn clause subset of the logic. In the next section the logic and the types of allowable rules will be extended to cater for negation and the possibility of evidence both for and against a proposition.

The normal situation in an expert system is a fixed body of rules (Horn clauses) which encodes the invariant knowledge about the problem at hand, and a set of facts which describes the current situation. The existence of a probabilistic logic allows some useful extensions to this view. For example, a fact can be stated as an additional "rule" of the form:

## dark-clouds $\Leftarrow \tau(1)$

and additionally uncertain evidence (I think there is a 50% chance that those are dark clouds) can be accomodated by a rule of the form:

## dark-clouds $\Leftarrow \tau(0.5)$

There is also nothing to stop the head of such a "factual" rule from from being an intermediate deduction which is also computed elsewhere by a set of rules. For example the user might say "it is going to rain with probability 0.5 although there is nothing in the rules to support this." and enter this as a "factual" rule:

rain  $\Leftarrow \tau(0.5)$ 

In order to effectively compute the consequences that result from such rule sets it seems to be necessary to restrict them to those with finite derivation trees (van Emden, 1986). That is, no recursive rules are permitted. This accords well with the normal situation in rule-based expert systems. The use of the  $\alpha$ - $\beta$  heuristic in such evaluations has been discussed for truth-functional logics of uncertainty (ibid), and can also be used here.

## Bit Sequences

Infinite sequences of bits can be made computationally tractable by approximating them with finite pseudo-random sequences of bits. The probabilities are in turn approximated by counting over these finite sequences. The major question this approximation raises is the accuracy with which the finite sequences represent the probabilities which would be generated by infinite length sequences. The accuracy is dependent on both the number of bits used and on the ability of the system to generate a large number of uncorrelated pseudo-random sequences. It is easily shown that the standard deviation of the estimated probability is given by  $\sqrt{p(1-p)/N}$ , where N is the number of bits and p is the probability of

a bit being true. So for N=32 and p=1/2 the results will be accurate to  $\pm 10\%$ , for N=1024 this is reduced to  $\pm 1.5\%$ . Even the  $\pm 10\%$  figure is well within the errors acceptable in existing systems (Shortliffe, 1976, p183). Conversely, 1024 bits is only 32 32-bit words so the logical operations required should not be too burdensome amongst all the other activities of an expert system. It is necessary to generate a number of uncorrelated random sequences for the different  $\tau$  sequences. As noted in (Gaines, 1969, Sec. 4.16) it is easy to generate a large number of uncorrelated sequences with p=1/2 using suitably long shift registers. For example, with N=1024 a single 33 bit shift register can deliver  $2^{23}$  independent sequences. These are readily combined to deliver sequences with appropriate probabilities. The stochastic computing systems described in (Gaines, 1969) are well suited to performing the types of computations needed here.

# NEGATION\_

Negation poses problems for Horn clause logics in general. The essential problem is that the statement of facts about particular situations must include information that some things are true, that others are false and that some are just not known. The usual way of handling this is to make the closed world assumption that *if something cannot be proven then it is false*. This is far too draconian for the current purposes as it makes it impossible to express that a fact is unknown. It is ridiculous to conclude that because I do not know whether it is sunny therefore it must definitely be cloudy.

A resolution of this is to extend the truth values to include *undefined* as well as true and false and to introduce two distinct forms of negation. As above, the truth values are infinite sequences (this will be ignored whenever convenient). The interpretation of true is "provably true" and of false is "provably false". The two forms of negation are denoted by  $\sim$  and  $\neg$ . Their truth tables are:

	t	u	f		t	u	f
~	f	u	t	-	f	t	t

~ should be interpreted as "provably not" and  $\neg$  as "not provable". So  $\neg$  corresponds to the normal closed world notion of negation. The truth tables for the various connectives are:

<u> </u>	t	u	f		t	u	<u>     f    </u>	Ę	=   t	u	f
t	t	u	f	t	t	t	t	t	t	u	f
u	u	u	f	u	t	u	u	u	t	u	u
f	f	f	f	f	t	u	f	f	t	t	t

Note that as a result of these definitions  $a \leftarrow b$  is equivalent to  $a \lor \neg b$ .

We would like the expression (x or not x) always to be true and (x and not x) always to be false. These need to be restated carefully in the new logic but there do indeed exist

statements with the correct properties. For example,  $x \lor \neg x$  is always true (x is provably true or not provably false) and  $\neg x \lor x$  is never false. Similarly,  $\neg(\neg x \land x)$ ,  $\neg(\neg x \land x)$  and  $\neg(\neg x \land x)$  are always true.

## Extended Horn Clauses

To accommodate these new notions the form of clauses allowed in the rule set can be extended as follows. The general form of rules is:

 $a \Leftarrow b_1, b_2, b_3, \dots b_n$ 

where a, the head of the clause, can be a term of the form x or  $\sim x$  and the  $b_i$  can be of the form  $\sim x$ ,  $\neg x$  or x where x is some atomic formula. Also, the  $b_i$  can be constants of the form  $\tau(p)$  or  $\sim \tau(p)$ .  $\neg x$  is not permitted in the head of a clause. Because the  $\tau$  constants can only occur on the right-hand side of rules which will not "fire" if any of their terms are undefined, there is no need to allow for undefined values in the infinite constants. There is no logical reason to exclude them, they just serve no useful purpose in this Horn clause logic.

The operational interpretation of these rules is that whenever x appears in the head of a rule whose body evaluates to true, then x is forced to true. Similarly if  $\sim$ x appears in the head of a rule then this forces the truth value of x to false. A weaker form of the closed world assumption is needed to completely define this procedure, that is, "if a value cannot be proven true or proven false then it is undefined". This seems much more palatable than the original form. These procedures ensure that none of the rules is provably false.

# Contradiction

This opens the possibility that two rules will attempt to force the same conclusion to be both true and false. In systems with a single truth value such a contradiction is catastrophic, the rule set has to be rejected (or debugged). If an infinite sequence of values is available the situation is not as bad. Any position along the sequence which generates a contradiction on any value at the head of a rule will cause all conclusions at that position to be ignored. In a rule set where this happens a lot, the precision of the inferred probabilities will drop as a smaller number of positions are available to count from. Although any high probability of contradiction will be untenable the system is at least graceful enough to allow some degree of contradiction and not fail catastrophically.

# ACQUISITION

# Deriving Constants From Observations

Putting logic aside briefly, we can consider the normal task of statistics which is exemplified by the question "predict whether it will rain given the current observations". Statistics tells us that what should be done is to count the number of times it has rained when the values of all the observations are the same as the current situation. The fraction of times that it has rained in these circumstances is the probability that it will rain again. However, typically there are many observations that can be made and it is unlikely that exactly the current situation has ever occurred in the past. This is now a form of the zero frequency problem, "what is the probability that something which has never occurred in the past will occur now". Statistics *per se* can give us no answer to this question; it must be sought from other knowledge about the problem.

One way to do this is to select some subpart of the current situation which is judged to be particularly relevant. If this subpart is sufficiently small then the current values will have occurred before and statistics can now be brought into play. What results is effectively a rule. For example, if the features judged to be relevant to rain are dark clouds and falling barometric pressure then statistics tell us how often the rule:

will-rain  $\Leftarrow$  dark-cloud and falling-pressure;

is true. More can be gained by reformulating this into a rule in probabilistic logic: will-rain  $\Leftarrow$  dark-cloud  $\land$  falling-pressure  $\land \tau_1(p)$ 

p can be estimated by counting all the observations on which the statistics are based. To be more precise the value of  $\tau_1$  can be computed using the inverse rules:

and

~  $\tau_1 \leftarrow$  dark-cloud  $\land$  falling-pressure  $\land$  ~ will-rain ;  $\tau_1 \leftarrow$  dark-cloud  $\land$  falling-pressure  $\land$  will-rain ;

The first of these is needed to avoid contradictions. The second is adopted to maximize the utility of the rule and to avoid the trivial situation where  $\tau_1$  is always false. The number of instances where  $\tau_1$  is (provably) true or false can be used to estimate p (the cases where it is undefined need not be counted).

These two rules for computing  $\tau_1$  are particularly useful because they are in the Horn clause form introduced earlier and so can be readily computed. Unfortunately, not all uses of logical constants can be so readily inverted. I will first consider some useful cases where the inversion is possible and then discuss some of the difficulties which arise in other cases.

The general case is that which arises when using nested rules of the form

will-rain  $\Leftarrow$  dark-cloud  $\land$  falling-pressure  $\land \tau_1$ will-rain  $\Leftarrow$  lightning  $\land \tau_2$ 

 $\leftarrow \tau_0$ 

Following the same reasoning as above the requirement that the rules fire as often as possible gives:

 $\tau_1 \wedge \tau_0 \Leftarrow \text{dark-cloud} \wedge \text{falling-pressure} \wedge \text{will-rain}$  $\tau_2 \wedge \tau_0 \Leftarrow \text{lightning} \wedge \text{will-rain}$ 

These two implications readily expand to the four Horn clauses

 $\tau_1 \leftarrow \text{dark-cloud} \land \hat{falling-pressure} \land \text{will-rain}$ 

 $\tau_2 \leftarrow \text{lightning} \land \text{will-rain}$ 

 $\tau_0 \Leftarrow \text{dark-cloud} \land \text{falling-pressure} \land \text{will-rain}$ 

 $\tau_0 \leftarrow \text{lightning} \land \text{will-rain}$ 

The requirement that the rules not be contradictory gives the two implications:

 $\sim(\tau_1 \wedge \tau_0) \Leftarrow \text{dark-cloud} \wedge \text{falling-pressure} \wedge \sim \text{will-rain}$  $\sim(\tau_2 \wedge \tau_0) \Leftarrow \text{lightning} \wedge \sim \text{will-rain}$  These cannot be so readily handled as there is ambiguity as to which of the constants should be set false when a rule fails. This ambiguity can be resolved by again saying that the rules should fire as often as possible and so only setting  $\tau_0$  false when both rules fail that is:

 $\begin{aligned} & \sim \tau_1 \Leftarrow \text{dark-cloud} \land \text{falling-pressure} \land \sim \text{will-rain} \\ & \sim \tau_2 \Leftarrow \text{lightning} \land \sim \text{will-rain} \\ & \sim \tau_0 \Leftarrow \sim \tau_1 \land \sim \tau_2 \end{aligned}$ 

Thus the original two Horn clauses give seven inverted Horn clauses for computing the probabilities of the constants.

This procedure is readily extended to any set of such nested rules where each constant appears only once. The rule sets may also be nested more than one deep as in:

$$\begin{pmatrix} a \Leftarrow b \land \tau_2 \\ c \Leftarrow d \land \tau_3 \end{pmatrix} \Leftarrow \tau_1 \\ e \Leftarrow f \land \tau_4 \end{pmatrix} \Leftarrow \tau_0$$

The relationships allowed by such rules are tractable in that their inverses give Horn clauses. They also seem to be able to encompass a useful set of inter-relationships between rules. It may be that a larger tractable class of rules is available but some simple examples will illustrate the problems involved. Consider the following three rules:

$$a \Leftarrow b \wedge \tau_1$$
$$a \Leftarrow c \wedge \tau_2$$
$$d \Leftarrow e \wedge \tau_1$$

The naive inverse rules for this case are:

i) 
$$\tau_1 \Leftarrow b \land a$$
  
ii)  $\neg \tau_1 \Leftarrow b \land \neg a$   
iii)  $\tau_2 \Leftarrow c \land a$   
iv)  $\neg \tau_2 \Leftarrow c \land \neg a$   
v)  $\tau_1 \Leftarrow e \land d$   
vi)  $\neg \tau_1 \Leftarrow e \land \neg d$ 

Because  $\tau_1$  occurs twice in the head of a rule it may be forced into a contradiction. For example, if a, b, and e are true and d is false then  $\tau_1$  should be set to both true and false. Presumably this is a signal that the rules should be debugged. However, the situation is slightly more complex than this. Because a occurs twice at the head of two of the original rules, i) and iii) together are too strong. If c is also true then the second rule will fire and it is not necessary to force  $\tau_1$  to be true. This situation can be resolved by the new weaker inverse rule:

i) 
$$\tau_1 \Leftarrow b \land a \land \neg \tau_2$$

Because of the very complex interelationships that can exist between the constants it is not clear in general how to resolve such situations and in particular how to guarantee that there are not circular realtionships in the inverse rules.

## Examples

A particularly interesting way of constructing the  $\tau$  constants is to use example cases provided by the expert. These are presumably carefully chosen synthetic cases which are important in practice. They can be handled in the same way as normal observations with the extra possibility that the counts can be more heavily weighted than cases encountered in day to day experience.

Observations where some of the facts are uncertain can also be handled in a similar way. For example, it may be known only that it is cloudy with probability 0.4 and sunny with probability 0.3 (leaving an uncertainty of 0.3). When cloudy is set true, one set of values for the constants will be computed; if it is set false (or undefined) then another set will be computed. So if a number of different pseudo-observations are made where cloudy is true 40% of the time and false 30% of the time then the counts for the various constants can be incremented by a suitably small value each time. That is the same procedure can be used for evaluating the inverse rules as is used for evaluating the original rules.

# SUMMARY

The major contribution of this paper is to provide a logical framework within which uncertain rules can be expressed and their uncertainties assessed from experience. The logic on which the rules are based does not suffer from many of the problems which other more *ad hoc* schemes are heir to. Most importantly the logic allows an automatic way of updating uncertainties including the correlations and redundancies between different rules.

There are a number of unresolved questions. It is not clear how easily such rules and their relationships can be expressed and understood by experts. Similarly, the subset of rule

types which can be easily inverted needs to be extended where possible and evaluated in practice.

# ACKNOWLEDGEMENTS

This work was supported by the Natural Sciences and Engineering Research Council of Canada.

# REFERENCES

van Emden, M.H. (1986) "Quantitative Deduction and its Fixpoint Theory," J. Logic Programming, 3(1) 37-53.

Gaines, B.R. (1969) "Stochastic Computing Systems," in Advances in Information Science Vol. II. Tou, J. (Ed). 37-173. Plenum.

Gaines, B.R. (1984) "Fundamentals of Decision," Studies in the Management Sciences, 20, 47-65.

Rescher, N. (1969) Many-Valued Logic. New York, McGraw-Hill.

Shapiro, E.Y. (1983) "Logic Programs with Uncertainties," in *Proc. 8th I.J.C.A.I.* Bundy, A. (Ed). 529-532. William Kaufman.

Shortliffe, E.H. (1976) Computer-Based Medical Consultations. New York, Elsevier.

Quinlan, J.R. (1983) "Inferno: a Cautious Approach to Uncertain Inference," *The Computer Journal*, 26(3) 255-269.

Zadeh, L.A. (1965) "Fuzzy Sets," Information and Control, 8, 338-353.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986 Cognitive Biases and Corrective Techniques: Proposals for Improving Elicitation Procedures for Knowledge-Based Systems

> By David A. Cleaves, Project Leader Fire Economics and Management Research USDA Forest Service Pacific Southwest Forest and Range Experiment Station 4955 Canyon Crest Drive, Riverside, CA 92507

Expert system output is only as good as the expert judgments on which the system is based. Expertise may be looked on as the ability to distinguish causal from random occurrence. Judgments of expert and novice alike have been shown to reflect systematic biases in comparison with normative statistical logic. These biases may be important where accuracy, consistency, and coherence are important attributes of the required judgment. Several biases and their cognitive origins are discussed in the context of building knowledge-based systems for wildland fire control. Preliminary guidelines are offered for recognizing and correcting biases during the knowledge elicitation.

#### INTRODUCTION

The elicitation of knowledge from experts is usually a bottleneck in the development of knowledge-based systems. It is a time-consuming process and is usually conducted in the absence of a systematic conceptual design. Few guidelines are available to help the knowledge engineer map out his/her course of inquiry to and pursue it efficiently. Improvements that are stumbled on in knowledge elicitation practice may escape recognition or may be difficult to transfer from one system-building experience to another.

One goal of knowledge elicitation should be to insure that the process is extracting the experts' best judgment because the programming phase of prototype development takes the quality of knowledge as given. This assumption should be constantly reexamined in light of two questions. What does "best" mean? Can we elicit rules and judgments that fully reflect what the expert knows?

The first question requires an understanding that expertise is multi-dimensional and is accessible through a rigorous assessment of expertise that are most important for a particular domain. Defining expertise is an important prerequisite not only in selecting experts but also in choosing approaches for knowledge elicitation. The proliferation of knowledge-based systems has created a need for flexibility in these definitions. The best expert for a particular task probably exhibits an optimum mix of performance in different dimensions. Different task domains require different mixes with emphasis on certain dimensions. Basically, a knowledge-based system should exhibit calibration, consistency, credibility, speed, understandability, and compatibility with the values of the individual or organization that uses it.

Maintaining the quality of expertise represented in any of these dimensions depends on fully capturing the best judgments and the important knowledge responsible for them. Often overlooked is how to detect, avoid, and correct

errors in judgment by the expert. Errors can occur not only in making an expert judgment, but also in explaining the underlying reasoning process. These errors can be found in the representation of decision rules and uncertainties in the prototype. For example, calibration, or the degree of agreement between a judged value (or relationship) and a real value, is one measure of expert performance. Lack of calibration can be caused by lack of knowledge or skill or from contextual and cognitive factors which prevent the expert from manifesting extant knowledge or skill on a particular problem. Training and experience may be the difference between the skill levels of the novice and expert. However, both may be subject to cognitive limitations. Error that results from these limitations will be called expert bias. Research comparing expert and novice performance has begun to identify knowledge differences, but expert bias has been relatively unexplored in expert systems development work.

Expert bias is a research topic in cognitive science and behavioral decision theory where the focus has been on subjective assessments of uncertain quantities and on perceptions of risk. Judgment research, as it is called, has produced many findings that could be useful in building knowledge elicitation systems or guidelines for manual interviewing techniques. The purpose of this paper is to review some of these findings and relate them to the knowledge elicitation process. The following questions will be addressed:

- 1. Are experts biased in their judgments?
- 2. Why do these biases occur?
- 3. Where would the knowledge engineer look for expert bias in the judgment process being elicited?
- 4. What might be done to correct or adjust for expert biases?

The research on judgmental bias is somewhat controversial within the field of psychology. Critics of the line of inquiry say that most biases have been discovered in experiments using unmotivated subjects and that implications drawn from such work have tended to overgeneralize. Not enough attention is given to individual differences. Bias is defined in terms of deviation from the normative logic of probabilistic reasoning, but fails to address the variety of other cognitive tasks to which statistical reasoning is poorly suited. Nevertheless, the evidence of bias in the form of different responses to the way these questions are framed or information is presented has been found in many replicated studies. The concept of expert bias is recognized by the public and decisionmakers. Associated questions of expert trust may be extremely important in future knowledge-based applications.

Why be concerned about expert biases? We need criteria for comparing and selecting expertise. Some experts with great knowledge may produce judgments so biased that better answers could be obtained from less "expert" subjects. The building of systems from multiple experts may need levels of calibration or measures on other dimensions of expertise on which to build portfolios of expertise to minimize the risk of bias. Expert bias may also be important in validating and evaluating expert systems. Better understanding of expert bias may enhance the ability to provide knowledge elicitation procedures and systems that improve, rather than merely mimic judgment processes. We may someday be able to test and correct for biases within the expert system itself, perhaps producing "purer" estimates on a consistent basis than the subject experts on which they are based.

Our perspective on these problems stem from our struggles in eliciting subjective assessment from experts in wildland fire control. Earlier efforts to model fire control processes involved decision analysis and real-time simulation, or fire gaming exercises. More recently, we have been developing three knowledge-based systems. The first seeks to estimate first and second order uncertainty of wildfire escapes on our controlled burn operations. The second system is designed to use multiple expertise to diagnose fire prevention problems in specific locations and to prescribe educational, enforcement, or other activities to address the problem. The third system is being designed to estimate how fast our crews and machinery can build fireline under various conditions of terrain complexity, fatigue, and other factors.

#### Are Experts Biased?

Bias can occur in any of the generic cognitive tasks which comprise problem solving. Fischoff (1984) describes four types of judgment tasks: (1) identifying active elements, (2) characterizing interrelationships, (3) assessing (estimating or predicting) parameter values, and (4) evaluating quality. Each of these tasks require different cognitive skills and are subject to different types of bias. In the expert system parlance, these tasks translate into (1) hypothesis and solution generation, (2) decision rule articulation, (3) assessing levels of uncertainty and confidence, and (4) hypothesis testing.

We might expect the expert to be better than the novice in all cognitive tasks. The defacto classification of expert is sometimes assigned to individuals solely on the basis of degrees and professional longevity, not necessarily on performance. However, we often do not know the conditions under which bonafide experts in actual decision settings provide information and reasoning or how to measure its reliability. Furthermore, the process of knowledge elicitation forces cognitive tasks into formats that may be unfamiliar to experts and novices alike. A prime example is probabilistic reasoning, assessing probabilities, as well as using probabilities or other uncertainty expressions in making inferences. The essence of expertise is to distinguish between causal and random occurrence. However it does not follow that the longer a person is exposed to chance processes, the better he/she will become at recognizing them. Research has shown basic differences between clinical and probabilistic reasoning in diagnosis but that probabilistic reasoning is often expected of many experts. Research (Winkler and Murphy, 1968; Kahneman, Slovic, and Tversky, 1982) has shown that experts in some fields do not produce any better estimates or forecasts in probabilistic terms than do relative novices. In addition, the expert is much less likely to say, "I don't know," which can have disastrous consequences in the decision

Expert bias that is defined for a particular dimension of expertise can be understood by the nature of the cognitive process. Popular topics of judgment research have been the degree and nature of calibration in the assessment of uncertain quantities. Additional effort has been directed at understanding human abilities in generating alternative hypotheses and problem solutions. Much could be borrowed from these research areas and developed into operational concepts for building valid knowledge-based systems. Although the concept of bias is based on some notion of a true or complete value, set of solutions, probability or relationship, the true value is either not known or perhaps revealed only after the expert judgment has been made. This often leads the knowledge engineer to prematurely accept those values produced by the subject expert or to validate them only against those of another "better" expert. This develops the tendency to embed limitations of human performance in expert systems without any knowledge of their influence on system performance.

Biases can occur in several ways (Table 1). Bias can be introduced in the method or the response format in which the judgment task is presented to the expert. For example, eliciting judgments as decision rules or probabilities may create bias if the expert doesn't really think about the problem in those terms. Conceptual bias results either from motivational processes, such as wishful thinking or other vested interests in the judgment's consequences, or from cognitive processes which are the gears of the expert's intuition. Motivational bias is very situational and can be difficult to detect and interpret without a strong familiarity with the organizational environment. At least some cognitive biases, on the other hand, tend to be systematic and can be recognized. The following discussion will focus on cognitive rather than task and motivational biases. Particular emphasis will be placed on the tasks of assessing uncertain quantities (probabilities and consequences). This is an important group of tasks because ultimately many quantities that enter into expert judgment are treated as uncertain quantities.

Table 1

Biases in Judgment

Task

Conceptual

Motivational

Cognitive

Biases in Judgment.

#### Why Do Biases Occur?

Biases result from the human's use of cognitive heuristics (Tversky and Kahneman 1974, 1981) that allow shortcuts through complex tasks. These heuristics should not be confused with the specific "if ... then" decision rules that are extracted during the knowledge elicitation process. Cognitive heuristics are metalevel modes of judgment which occur outside the awareness of the individual, but which drive reasoning and judgment. Described below are several major cognitive heuristics which have been observed in experimental and real-life judgments.

Anchoring: Individuals tend to search for a first approximation or natural starting point and then to adjust judgments of subsequent outcomes or probabilities using the initial estimate as a base. Available information

which would support extreme adjustments is seldom utilized (Tversky and Kahneman, 1974).

Availability: Events, quantities, or relationships that are more easily brought to mind are more apt to be judged as likely. Unlike purely probabilistic reasoning, frequency is not the only determinant of how psychologically available an event may be. Familiarity, saliency, and imaginability strongly enhance the retrievability of an event, even though the event itself may be rare. Overreliance on availability may lead to systematic disagreement with more objectively determined estimates of likelihood. Very recent experiences and observations or those that are extremely vivid or memorable have inordinate weight in expert responses, preventing the expert from fully relying on the depth of long years of experience. Associated biases include the preference of concrete over abstract information, extreme sensitivity to present conditions, and illusory correlation among various components and factors in a problem (Tversky and Kahneman, 1973).

Representativeness: Likelihoods of events and risks are judged by their degree of similarity to familiar or stereotypic events rather than statistical frequency. The human tends to extrapolate occurrences from a small sample of events to what he/she perceives to be a similar, larger class of events. Even those experts with deep knowledge of the subject area tend to predict outcomes that are most representative of the evidence at hand with relative insensitivity to its reliability or to what is generally known about prior probabilities (Kahneman and Tversky, 1973). Representativeness can result in stereotyping of outcomes, e.g., imagining that a set of conditions will result in certain consequences because it exhibits "classic" characteristics. Bias results when objective information is forsaken for a more narrowly defined set of these characteristics. The more detailed a scenario is, the more likely it seems, even though the combination of characteristics in an actual scenario is less likely to actually occur.

Internal Coherence: Information or judgments consistent with an individual's previous beliefs are favored over those that are less consistent or suggest the need for additional testing. For example, the plausibility of a scenario or an estimate of variability becomes more heavily weighted in its perceived likelihood than its probability of occurrence. This effect is especially pernicious when the expert has expressed the same judgment in the past even without validation of his/her prediction. Experts may give short shrift to conflicting evidence or scenarios outside their range of previous experience and approval.

**Consistency:** Consistency of information sources can lead to increases in the confidence in judgment, but not to increased predictive accuracy or other performance. A small body of consistent data has much greater effect than a larger body of less consistent data, especially where the task (e.g., planning) is nonrepetitive and there is little feedback on the outcomes (Hogarth, 1980). People may actually further collect information that is biased toward initial preferences in order to increase the consistency.

#### Where in the Judgment Process Do Biases Occur?

Cognitive heuristics do not uniformly produce poor judgments. Indeed, these are natural processes with which the expert develops his/her clinical

abilities. However, excessive reliance on certain heuristics can result in systematic biases (Hogarth and Makridakis, 1981, Kahneman et al., 1982). Different cognitive tasks are prone to different types of biases (Table 2). These biases have been identified in experimental and operational contexts.

Та	ble 2								
Hypothesis and solution generation			Decision rule articulation	Uncertainty assessment			Hypothesis evaluation		
-		0	attribution errors	0	overconfidence	0	selection bias		
0	option generation deficiency	0	illusory correlation	0	hindsight	0	information		
		0	order effects	0	conservatism				
		ο	information bias			0	order effect		
0	selection bias	0	"law of small numbers"	0	rare, conjunctive event bias	0	hindsight		
				0	gambler's fallacy	0	overestimation of completenes		
				0	regression bias				
				0	base rate ignorance				

Cognitive Biases in Expert Judgment Tasks.

Hypothesis and Solution Generation

**Option Generation Deficiency:** Omitting important scenarios or elements of in retrospect scenarios. Our fire experts complain most bitterly about scenarios that were totally unforseen during their planning process. Group simulation techniques have somewhat alleviated this, but the problem is still acute where single experts are unaided by divergent perspectives on a problem.

Selection: Limiting selection to only options that the individual has experienced or expects to occur. Our experts have gained knowledge from experience in local areas for which our systems are being built, as well as experience gained from service in other geographical settings. Some base their judgments almost entirely on local experience and tend to have great difficulty translating their experience from other settings.

#### Decision Rule Articulation

Attribution Errors: Attributing too much responsibility to one condition or factor; attributing success to skill and failure to chance, or vice versa. Human error and the lack of proper firefighting resources are the two factors influencing fire escapes that are most quickly cited by our experts. Without prompting, they often fail to consider the influence of a variety of other environmental conditions such as wind, air temperature, and the arrangement of vegetative fuels.

Illusory Correlation: Belief that two or more variables co-vary when in fact they do not or that patterns exist in what is actually a random process. Changes in fire occurrence patterns are especially difficult to diagnose. There is a tendency for the fire prevention person to see causal patterns that are not there, simply because fire incidence may increase at the same time certain human activities do.

**Order Effects:** Assigning undue importance to the first items in a sequential presentation of conditions (primacy), or to the last items (recency). Because of the dynamics of wildland fire, firemen learn to treat the most recent information as the most valuable, whether or not it is more reliable. In the knowledge elicitation context, they respond strongly to new pieces of information that may be, on further questioning, unimportant.

**Information Bias:** Overweighting conditions described by information that is concrete, absolute, and consistent with initial inferences while underweighting conditions with abstract, relative, and conflicting evidence.

Law of Small Numbers: Mistakenly expecting small samples to be highly representative of a population, leading to undue confidence in early results, premature truncation of the search process, and little appreciation for variability in the sample of conditions given.

#### Assessment of Uncertainty and Confidence

**Overconfidence:** Estimating much higher discrete probabilities (levels of certainty) or much tighter distributions (and intervals) than the actual. Assessed probabilities of fire escape and distributions of fireline production rates and escape levels are almost always higher and tighter than are shown in data. Typically, our human assessors are keying their judgments on representativeness to recent or vividly remembered catastrophes.

**Hindsight:** Overestimating the probability of events just because they have occurred in the past. The mistaken belief that an event was inevitable leads to undervaluing new facts and research. Experts who have a long history in one narrowly circumscribed area often weave their entire assessment around the range in events they have personally experienced.

**Conservatism:** Failure to revise an opinion on receipt of new information. Once a judgment is made, some of our experts are extremely stubborn about even considering new information. In their defense is the fact that survival has depended on decisiveness and avoidance of constant revision. It is difficult, however, for the knowledge engineer to determine where and when the initial

judgment is made. Important information that is delivered late in the assessment task is often given short shrift or completely ignored.

Rare and Conjunctive Events: Overestimation of the probability of infrequent single events or the joint probability of independent events.

Gambler's Fallacy: Overestimation of the probability of one event or value when an unexpected number of other events or values have occurred. Strings of escapes or "hot streaks" (successes) are not often predicted although they may be just as probable as the escape/no escape sequence.

**Regression Bias:** Using extreme values of a variable to predict extreme values of the next observation (failing to allow for regression to the mean). This can be especially active in sequential judgments. High performance is often followed by lower performance even in the presence reward, merely because of the random element, but individuals often mistakenly assume it is cause-and-effect.

Base Rate Effects: Relying exclusively on specific, descriptive information and neglecting prior probabilities (base rates) of the general class of events or values.

#### Hypothesis Testing

Note: The process of testing hypotheses is subject to many of the biases described above under other cognitive tasks. Most important are probably:

- o overestimation of completeness
- o order effects
- o selection
- o hindsight
- o information bias

## Monitoring and Corrective Procedures

Corrective procedures have been designed to combat biases in individual and group judgment (Table 3). Our research is aimed at bringing valid procedures into the realm of expert systems knowledge elicitation. Mechanical procedures manipulate the task or adjust the judgments after elicitation. Behavioral procedures use interviewing and group interaction techniques to encourage full employment of the assessor(s)' knowledge. Most techniques have been developed to address particular biases, so the knowledge engineer should have some a priori notion of what types of biases may be more active at various stages in the judgment task that is being modeled. Not all procedures have been subjected to rigorous testing in knowledge elicitation practice, so caution and exposure to a wide variety of techniques is urged. The literature on these procedures is extensive, although the references cited herein provide a good introduction. Table 3

## Mechanical

<u>Task</u>

Assessment

visual props scoring rules

response mode consensus weighting\*

interaction\*

\*Applies to group assessment.

Corrective Procedures for Decision Rule and Uncertainty Assessment Biases.

Behavioral

decomposition

logic challenges

focusing

training

consensus

Described below are some classes of corrective techniques which have been applied to the tasks of articulating decision rules or uncertainty levels. This listing is not an exhaustive comparison of techniques, but rather a taxonomy to guide the interested knowledge engineer. This list does not address techniques designed to improve hypothesis generation and selection.

#### Mechanical Techniques

Visual Props: Letting the individual select, create, and compare visual patterns rather than verbal expressions to represent his judgment. An example is the probability wheel (Spetzler and Stael Von Holstein 1975) which allows the respondent to work with the concept of probability without having to consciously wrestle with the numerical intricacies.

Response Mode Changes: Varying the format of the requested judgment, hypothesis, or uncertainty level can identify inconsistencies and find more natural means of expression. For example, rare event assessments are less biased when they are elicited as odds (l in 1000) than when they are elicited in probabilities or decimals (.001). People can more easily express larger probabilities and quantities in an indirect mode by comparing or ranking values. Indirect mode has been shown to be less biased than the direct expression of each assessment, say in physical units. Also, asking for the likelihood of a given value usually produces better calibrated answers than asking for the value for a given likelihood level.

Scoring Rules: Using some function of the difference between actual and assessed values either to motivate individuals or to recalibrate assessments expost facto.

**Consensus Weighting:** Combining judgments from several individuals by creating weighted means of their responses on the same judgment task. Weighting bases may include differences in technical ability, self-rating of confidence, prior calibration, or other factors.

### Behavioral Techniques

Focusing: Structuring both the task and the interviewing environment so that specific biases are identified and corrected as they become symptomatic. One focusing technique is providing "reminder" information on base rates and requiring individuals to respond as to how they assimilated this information into their assessments.

**Decomposition:** Breaking down the quantity or relationship into subevents or important factors, each of which may be more probable and easier to assess than the original composite. These subcomponents can be weighed individually for their contribution to the overall strength of relationship or likelihood. Assessment of rare or conjunctive uncertainties have been shown amenable to this approach.

**Training:** Individuals can sometimes remove bias if they understand and can manipulate the representation format (certainty factor, production rule, frames, etc., themselves). Training in probability assessment has been successful in improving calibration, especially when feedback on performance was frequent, immediate, and specific to the task.

**Logic Challenges:** Exhorting individuals to justify reasons (and to offer opposing explanations or assessments) for their own answers. Biases are sometimes revealed when an individual must list as many reasons for as those against a (higher) best guess. Answers often change in response to this upfront checking.

**Consensus Interaction:** Presenting, in a group setting, each individual with feedback from the other members about his/her judgments. When the level of controversy is controlled by the knowledge engineer, this can encourage reconsideration and better judgments. The degree of face-to-face interaction distinguishes the various techniques in this class. The two best examples are Delphi and nominal group techniques. There is no best technique, but a commonly recommended approach consists of nominal group application for initial estimates, structured feedback to the group, and silent voting on two or three estimates. This estimate-talk-estimate sequence works better than other sequences for most tasks.

## CONCLUSIONS

Expert judgment is human judgment and as such can be improved. Effective knowledge elicitation should incorporate strategies for ensuring that the best judgments are passed on to the knowledge representation stage. Unless this is done, expert systems will perpetuate and perhaps exacerbate some of the limitations of human judgment which have been recognized in research on cognition. Recognizing potential flaws in judgment requires the knowledge engineer to be familiar with the domain and to develop explicit performance and selection criteria on the various dimensions of expertise. Some dimensions may warrant more monitoring and debiasing than others.

Research on judgment has tested concepts and techniques for correcting judgmental bias. Some of this work has used experts as subjects. Surprisingly little has been done to apply these approaches in knowledge elicitation practice, much less to integrate them into a framework for interviewing or designing knowledge elicitation tools. The cognitive heuristics that have been identified in judgment research also appear to be the very mechanisms through which expert performance is distinguished. Paradoxically, the same processes may be responsible for expert performance and common sense bias. Little research has been done to determine what conditions under which this paradox occurs. Using a model of judgment to guide the knowledge elicitation process could help to identify where biases may lurk and during the codification of knowledge structures.

During subsequent rounds of prototyping and elicitation, these biases could be sought out with questions and corrective protocols. The utility of expert systems could be enhanced through the development of enhanced or "leveraged" intuition. Systems developed on experts for experts could incorporate interactive procedures for checking and correcting biases and inconsistencies in judgmental inputs. Continued use of such systems in advisory roles or as training devices could actually improve expert judgment in some domains.

We are testing various approaches for identifying and correcting biases in our knowledge elicitation efforts. Our goal is to develop a set of general guidelines for knowledge engineers to use in manual or automated elicitation. Preliminary observations of our own knowledge elicitation experiences have pointed out the need to separate the knowledge elicitation and representation phases and to delay prototyping until possible biases and knowledge structures can be explored fully. Premature prototyping captures biases that may be difficult for the expert or some truth-maintenance systems to detect.

Open-ended interviewing should be mixed with more highly structured styles as the process converges on specific judgments. For example, overuse of anchoring can be combated by asking the expert for extreme judgments before most of his/her likely ones. Overreliance on representativeness appears to be partially corrected by providing the expert with base-rate information and actively seeking verbal consideration of it. Many scenarios should be presented, including those with inconsistent information. Care should be taken not to let classic or well-publicized incidents or scenarios become prominent. Confidence factor assessments should not be accepted without close scrutiny. They can act as guides for more indepth questioning, but high confidence levels can hide as many problems as low ones turn up, depending on the expert's ability to introspect.

#### LITERATURE CITED

Fischoff, B., 1983. Judgmental aspects of risk analysis. Decision Research, Eugene, OR. 42 pp.

Fischoff, B., 1984. Eliciting information from experts. In: Research Needs for Human Factors, pp. 33-48, Washington, D.C., National Academy Press.

Hogarth, R. M., 1980. Judgement and Choice: Strategies for Decision. New York, Wiley.

Hogarth, R. M. and Makridakis, S., 1981. Forecasting and planning-an evaluation. Management Science 27(2):115-138.

Kahneman, D. and Tversky, A., 1973. On the psychology of prediction. Psychological Review 80:751-757.

Kahneman, D., Slovic, P. and Tversky, A., 1982. Judgement Under Uncertainty: Heuristics and Biases, Cambridge, Cambridge University Press, 555 pp.

Spetzler, C. S. and Stael Von Holstein, C. S., 1975. Probability encoding in decision analysis. Management Science 22(3):340-358.

Tversky, A. and Kahneman, D., 1973. Availability: a heuristic for judging frequency and probability. Cognitive Psychology 5:207-232. Tversky A. and Kahneman, D., 1974. Judgments under uncertainty: heuristics

and biases. Science 185:1124-1131.

Tversky, A. and Kahneman, D., 1981. The framing of decisions and the psychology of choice. Science 211:453-458.

Winkler, R. L. and Murphy, A. H., 1968. "Good" probability assessors. Journal of Applied Meteorology 7:751-757.

## The Application of Psychological Scaling Techniques to Knowledge Elicitation for Knowledge-Based Systems

Nancy M. Cooke and James E. McDonald Computing Research Laboratory and Department of Psychology New Mexico State University Las Cruces, NM 88003 USA

## ABSTRACT

A formal knowledge elicitation methodology that incorporates psychological scaling techniques to produce empirically-derived knowledge representations is discussed. The methodology has been successfully applied in several domains and overcomes many of the difficulties of traditional knowledge elicitation techniques. Research issues pertaining to the use of scaling techniques as knowledge elicitation tools are outlined and a particular issue, the elicitation of levels of abstraction in knowledge representations, is discussed in detail. Results from a study on the elicitation of knowledge about levels of abstraction for a set of Unix commands from experienced Unix users indicated that the representations obtained using this methodology can be used to obtain more abstract (i.e., categorical) representations of that knowledge.

## INTRODUCTION

Currently, the process of eliciting knowledge from human experts for the purpose of transferring this knowledge to an expert, or knowledge-based, system suffers from numerous problems. Not only is the process time consuming and tedious, but the weak methods currently used (i.e., interviews and protocol analysis) are inadequate for eliciting tacit knowledge and may, in fact, lead to inaccuracies in the knowledge-base. Much expert knowledge is not available to conscious introspection (i.e., it is automatic or compiled) and consequently, experts may give erroneous or incomplete accounts of their knowledge (Nisbett & Wilson, 1977). Indeed, Ericsson and Simon (1984) contend that information must be in the focus of attention (i.e., short-term memory) in order to be verbalized. One approach to these problems has been to formalize interview and protocol analysis techniques (Ericsson & Simon, 1984; Hoffman, 1986). Alternatively, several investigators have suggested that psychological scaling techniques, such as cluster analysis and multidimensional scaling, be used as a means of eliciting knowledge from domain experts for the purpose of building expert systems (Butler & Corter, 1986; Cooke & McDonald, 1986; Gammack & Young, 1985). It is generally believed that these techniques are more objective and formal than traditional knowledge elicitation methods.

We (Cooke & McDonald, 1986) have proposed the development of a tool kit of scaling techniques that could be used to elicit and represent core components of knowledge from an expert or group of experts. The core knowledge-base would consist of basic concepts, relations, facts, rules, and procedures that are at the heart of the domain and could be used to either guide subsequent knowledge elicitation (i.e., interviews, protocol analysis, or more scaling techniques) or act as a starting point for an automated system which continually refined the knowledgebase.<sup>1</sup> In the following sections we discuss some of the scaling techniques comprising the tool kit.

## SCALING TECHNIQUES

There are several psychological scaling techniques that produce structural representations of knowledge based on human judgments. All of the scaling techniques discussed here require as input a matrix of distance estimates for all pairs of items in a set to be scaled. The techniques differ largely in the types of structural representations they produce. For instance, multidimensional scaling procedures (e.g., Kruskal, 1977; Kruskal & Wish, 1978; Shepard, 1962a, 1962b) generate spatial representations in which concepts are located in an *n*-dimensional space, where each dimension might represent a particular feature (e.g., size, ferocity, predacity). Alternatively, hierarchical cluster analyses (e.g., Johnson, 1967) produce taxonomic representations. Whereas hierarchical representations might be adequate for representing some types of knowledge, they are inadequate for representing knowledge that is not strictly hierarchical in nature. Recently, a network scaling technique (Pathfinder: Schvaneveldt & Durso, 1981; Schvaneveldt, Durso, & Dearholt, 1985) has been developed that produces graphs which are not constrained to hierarchical relationships.

### Multidimensional Scaling

Multidimensional scaling (MDS) is an elaborate least-squares fitting technique. Most MDS programs require that the user input a symmetrical distance matrix and specify the number of dimensions to which the data should be fit. The algorithm then attempts to fit the data to the specified number of dimensions by minimizing the amount of "stress," or deviation from the optimal fit, in an iterative fashion. There are several types of MDS analyses available, the most common variables being whether they are metric or nonmetric (i.e., whether they assume interval or ordinal data) and weighted or nonweighted (i.e., whether individual subject matrices are considered or only a single composite matrix). In general, nonmetric techniques are to be preferred to metric, since it is often difficult to insure that the data have interval properties. Also weighted techniques are more robust than nonweighted (i.e., they tend to reduce the problems associated with local minima and are claimed to be nonrotatable). The MDS solution consists of a set of dimensionless points (one for each item in the set) in an ndimensional space. While the relative positions of items can be considered optimal, the orientation of the space itself is arbitrary (i.e., it is rotatable). Thus, it is often difficult to identify the dimensions (if they are in fact meaningful) or to determine the correct number of dimensions to use.

<sup>&</sup>lt;sup>1</sup>We believe that it is important to distinguish knowledge elicitation (the topic of this paper) from knowledge acquisition. The latter term is quite general and often ambiguous, but in the context of artificial intelligence it most often refers to machine learning. The machine learning approach is one possible solution to the knowledge engineering bottleneck which occurs when human knowledge is to be transferred to a machine. An alternative solution is to improve upon the techniques available for extracting knowledge from human experts. We use the term knowledge elicitation to refer to this approach. We feel that the two approaches complement each other in many ways. Knowledge elicitation techniques could be used to produce the core knowledge-base and knowledge acquisition techniques could be used to augment and refine this core knowledge. One can think of knowledge elicitation as a special case of learning, that is, learning by being told.

## Cluster Analysis

There are numerous clustering techniques (e.g., hierarchical, overlapping, disjoint), but hierarchical clustering schemes (HCS) are the most common. Furthermore, there are several HCSs, the most common being the minimum (connected) and maximum (diameter) methods. HCS is conceptually simple and consists of repetitively combining the two closest objects in the data matrix into a single cluster. Once formed, distances to this new cluster must be computed and can either be the minimum or maximum of the distances from each of the objects (the average is also occasionally used). The output of the HCS is a "tree" with connections between items and clusters at various levels. The number of clusters ranges from n (each item is a separate cluster) to 1 (all items in a single cluster). One of the difficulties in employing HCS for knowledge representation is deciding on an appropriate cutoff level, that is, when to stop grouping items together. Unfortunately, this decision must be left to experimenter judgment.

## Pathfinder

The networks generated by Pathfinder consist of nodes representing concepts (or objects or actions), and links between some pairs of nodes, representing relationships between those concepts. Pathfinder, like other scaling techniques, requires estimates of relatedness or distance for all possible pairs of concepts in a set. In Pathfinder these concepts are to be represented in the network as nodes. Basically, the Pathfinder algorithm determines whether or not a link will be present between each pair of concepts. Links are assigned weights (distances in the data matrix) corresponding to their strength. A link is added if the minimum distance between the concepts based on all existing paths (chains of one or more links) is greater than the distance estimate for that pair.

Pathfinder allows for systematic variation in the complexity (number of links) of the resulting networks as parameters of the method are varied (i.e., r and q). The parameter r is based on the Minkowski r-metric. Its value can range from 1 to  $\infty$ and determines how path length is computed. While continuously variable, simple interpretations exist only for r equal to 1 (all links in the path are added), 2 (the Euclidean distance is computed), and  $\infty$  (the longest link in the path is computed). The q parameter determines the maximum number of links that can be included in a path. When  $r = \infty$  and q = n - 1 the network is minimally connected and the data are assumed to be ordinal. For these reasons,  $r = \infty$ , q = n - 1 are generally the preferred values for these parameters. Regardless of the values specified for r and q, the resulting network solution is inherently not dimensional and is difficult to represent graphically. While nodes and connecting links are specified, no real limitations are placed on the location of nodes in the n-dimensional space. Further, links are at most weighted arcs, since no information about the meaning of links is supplied as part of the analysis.

## Applications of Scaling Techniques

In general, scaling techniques such as those described above have been used in cognitive psychology to study the organization of concepts in memory (e.g., Rips, Shoben, & Smith, 1973; Shoben, 1983). Cooke, Durso, and Schvaneveldt (1986) found that lists of words organized according to either the Pathfinder or multidimensional scaling representations of those concepts were easier to learn than unorganized lists. Further, they found that a list organized according to the Pathfinder representation was easier to learn than a list organized according to the multidimensional scaling representation. These results provided evidence for the psychological validity of scaling representations in that the representations were predictive of learning and recall. Scaling techniques have also been used to investigate cognitive structures underlying human expertise in domains such as air-to-air flight maneuvers (Schvaneveldt, Durso, Goldsmith, Breen, Cooke, Tucker, & DeMaio, 1985) and computer programming (Cooke and Schvaneveldt, 1986). Information gained from such studies can be applied to design, training, and selection problems. For example, Cooke and Schvaneveldt (1986) had expert, intermediate, novice, and naive computer programmers judge the relatedness of all possible pairs of a set of abstract programming concepts (e.g., *algorithm*, *global variable*, *debug*). These data were used to empirically derive Pathfinder network representations of programmer knowledge of these concepts. In Figures 1 and 2 the average naive and expert programmer network representations are displayed. Results indicated that the representations varied depending on programming experience and that the changes were systematic with expertise, providing evidence for evolution of the structures. The networks also revealed some critical associations of expert programmers and some misconceptions of naive programmers.







Figure 2. Pathfinder network representation of the average cognitive structure of expert programmers.

In addition, scaling techniques have been applied to the design of the humancomputer interface (McDonald, Dayton, & McDonald, in preparation; McDonald, Dearholt, Paap, & Schvaneveldt, 1986; McDonald, Stone, Liebelt, & Karat, 1982; Roske-Hofstrand & Paap, in press; Tullis, 1985). McDonald, Dearholt, et. al. have proposed a formal interface design methodology based on user knowledge. One of the applications they discussed involved the development of an indexing aid for the Unix on-line documentation system (the "man" system). Their approach was to empirically derive a model of the Unix operating system from experienced users of the system, then to base the design of the index on this model. During the model-building phase of their study, McDonald, Dearholt, et al. (1986) first used a sorting methodology to obtain estimates of distance for approximately 200 basic Unix commands (section 1 of the man system) from 15 experienced Unix users. They then performed Pathfinder and cluster analyses on the obtained proximity matrix. These analyses formed a base-level Unix user's model. Some of the specific results of this study are summarized later in this paper.

## Advantages of Scaling Techniques as Knowledge Elicitation Tools

As previously mentioned, scaling techniques can overcome many of the criticisms of knowledge elicitation techniques such as interviews and protocol analysis. In addition to facilitating knowledge elicitation, scaling techniques provide information concerning the structure of knowledge that might serve as a basis for representing that knowledge in the system. For example, the organization inherent in the scaling representation might suggest an organization of a set of production rules that would facilitate search. More generally, we believe that our knowledge elicitation methodology will allow the development of empiricallyderived knowledge-bases in which the information elicited determines the content and architecture of the system, as opposed to the traditional approach in which the elicitation of knowledge is guided by an existing or proposed system architecture.

Traditionally, knowledge elicitation methodologies have involved interviews with one or more experts. The use of multiple experts has tended to increase the opportunity for conflicts in the knowledge-base, which might simply be a reflection of the fact that there are several different, yet correct, solutions to problems. However, it is not clear how idiosyncrasies among different experts should be coded in a knowledge-base. One advantage to the use of scaling techniques is that they can be applied to group, as well as individual data. The resulting group solution can be thought of as a prototypical representation of expert knowledge. Thus, scaling techniques also provide a means of combining knowledge from multiple experts.

In summary, the scaling approach to knowledge elicitation has a number of advantages over traditional knowledge elicitation techniques. Psychological scaling techniques require less introspection on the part of the expert than traditional interviews in which the expert is asked, "tell me what you know and how you Scaling techniques require experts to make judgments rather to know it." introspect and verbalize about how they do what they do. Ericsson and Simon (1984) have noted that appropriate cues are necessary in order to retrieve knowledge from long-term memory. Consequently, some of the problems of verbal reports might be avoided by providing experts with specific cues in the form of the concept set. Scaling techniques also offer a more formal, as well as more and objective, means of eliciting knowledge than traditional techniques. Scaling solutions are standard, in the sense that given a set of distance estimates and a specific scaling technique two different researchers/knowledge engineers will independently produce identical representations. On the other hand, the same two individuals may have radically different interpretations of the same protocol. Finally, the scaling approach can handle, and is in fact well-suited to, elicitation of knowledge from multiple experts.

## RESEARCH ISSUES CONCERNING THE USE OF SCALING TECH-NIQUES AS KNOWLEDGE ELICITATION TOOLS

Whereas there are many foreseeable advantages to using scaling techniques for knowledge elicitation, the application to this particular domain raises several methodological issues. In general, the scaling techniques themselves are wellspecified, but there are other aspects of the methodology as a whole (e.g., method of data collection) that require investigation. Thus, the issues outlined in this section are aimed at formalizing the methodology from start to finish. By formalizing all aspects of the methodology, its application will be simplified and subject to less interpretation on the part of the researcher/knowledge engineer.<sup>2</sup> Recently, there have been several studies conducted that have addressed one or more of the research issues. In fact, the development of a knowledge elicitation tool kit can
be thought of as a series of studies designed to answer questions and solve problems related to the application of scaling techniques to knowledge elicitation. In the following sections the major issues are presented, along with a brief summary of research directed at each particular issue. The issues are organized according to the three major phases involved in scaling analysis: 1) data collection, 2) scaling, and 3) interpretation. It should be emphasized that the fact that clearly delineated phases and research issues can be identified attests to the formal qualities already present in this approach.

## **Data Collection**

All of the scaling techniques discussed require as input a matrix of distance estimates for all pairs of items to be scaled. It is assumed that a relatedness estimate for a pair of items corresponds to the distance between the corresponding concepts in memory. According to the distance metaphor of memory, concepts that are semantically related are closer in memory than concepts that are less related. The scaling solutions all represent the notion of distance in different ways. Two concepts that are judged to be highly related will generally be close in the spatial solution of MDS, appear in the same cluster in cluster analysis, or be joined by a link in a Pathfinder network. The four issues that follow have to do with the collection of distance estimates for a set of items.

Obtaining a set of items to be scaled. The first step in the data collection phase of these techniques involves the selection of a set of items (concepts, objects, actions) to be represented. It is usually appropriate to limit the number of items in the set in order to simplify data collection and interpretation of the resulting solution. Therefore, it is important to avoid including items that are irrelevant to the domain and at the same time attempt to capture all (or at least most) of the critical items of a domain. Unfortunately, no satisfactory techniques exist for identifying the important items in a domain or for selecting a subset of the important items once they have been identified. In some cases a predefined set of items exists (e.g., the set of all Unix commands), however, the set may be inappropriate. For instance, McDonald, Dearholt, et al. (1986) found in their study of the Unix system that only 44 of the 219 commands in the set were known by all of the experienced users. In addition, event records of command usage collected from experienced Unix users indicated that out of 235 different commands used, only 112 were from the original set of 219. Typically, researchers select a set of items based on their intuitions about the domain or the intuitions of a domain expert. Because it is often difficult to introspect and to verbally express mental processes (Nisbett & Wilson, 1977), this set of intuitively derived items may be incomplete or inappropriate.

Research has been conducted on various methods for eliciting domain-related information from experts (Cooke & McDonald, 1986). The goal of this work was to develop a methodology for identifying the core set of basic ideas in a particular domain. It should be noted that this goal differs from the general goal of knowledge elicitation in which not only ideas are needed, but also relations, rules, and facts connecting the ideas. Cooke and McDonald (1986) investigated four idea elicitation tasks that were given to domain experts: 1) listing critical domain concepts, 2) listing steps involved in a domain-related task, 3) listing chapter titles and subtitles for a hypothetical book on the domain, and 4) extracting critical ideas from an interview with an expert. Results indicated that the four

<sup>&</sup>lt;sup>2</sup>It should be pointed out that many of these issues are relevant for other applications as well (e.g., interface design and studies investigating expertise).

methods differed in terms of the number of ideas that were generated as well as the types of knowledge (concepts, facts, general rules) generated. Interestingly, less than ten percent of the ideas took the form of "if-then" rules, suggesting that knowledge is not naturally conveyed in this form.

**Contextual effects.** The selection of an appropriate set of items is important for several reasons. An obvious ramification of selecting an incorrect or incomplete set of concepts is a resulting knowledge representation that is also incorrect or incomplete. However, there is another not-so-obvious reason that the set of items plays an important role. The distance estimates supplied by the subjects are affected by the context or frame supplied by the rating environment (Tversky, 1977; Murphy & Medin, 1985). Murphy and Medin (1985) have argued that theories underly similarity judgments and that different theories are appropriate in different contexts. Consequently, context changes, such as subtle differences in instructions, can affect relatedness judgments. For example, Murphy and Medin (1985) suggested that judgments of relatedness between concepts such as *chil*dren, jewelry, and photograph albums would change given the context of a burning house. Like instructions, the specific concepts that make up the concept set provide a context for relatedness judgments. For example, penguin is very similar to bird when the other concepts are types of vegetables, but not so similar when the other concepts are types of birds. Therefore, the particular concepts chosen to make up the concept set have nontrivial consequences for the outcome of the psychological scaling representations. Research is being conducted that investigates the effects of different contexts on the scaling representation of a set of items (Roske-Hofstrand & Paap, 1986). Results thus far have indicated that relatedness estimates are indeed affected by the addition or deletion of items in a given set.

Methods of collecting distance estimates. Distance estimates between all possible pairs of items can take the form of a matrix in which the rows and columns designate specific items in the set. Such matrices may be symmetrical (i.e., the distance from A to B is equal to the distance from B to A) or asymmetrical (i.e., the distance from A to B is not equal to the distance from B to A), although none of the current scaling techniques except Pathfinder make use of this information.

Distance estimates can be obtained through paired comparisons, sorting, listing, and event record analysis. In the paired-comparison task judges are required to rate the relatedness or similarity of all n(n-1)/2 pairs of items. Distance estimates are computed as the inverse of the relatedness ratings. Theoretically, asymmetrical distances can be obtained using this technique, but it is a time consuming task — extremely so for large sets of items.

Sorting entails having judges sort items into piles based on shared relationships. For a group of judges, distance estimates are the inverse of the frequency with which pairs of items are placed in the same pile. This technique is considerably less time consuming than the method of paired comparison.

Distance estimates can be obtained from lists of relevant domain concepts by computing the conditional probability that one item follows another in the lists. However, this technique does not insure that estimates of distance for all items of interest are obtained. Alternative methods for computing distance from lists are available, for instance, number of intervening items in the list. Similar techniques can be applied to event records where the lists consist of actions recorded as experts perform domain-related tasks.

Research is needed that compares these various data collection methods in terms of the distance estimates that they produce. Representations may differ depending on the method used to obtain the distance estimates. The advantages and disadvantages of each method need to be addressed explicitly and the appropriate applications for a given method should be identified. As a preliminary step toward resolving this issue, Roske-Hofstrand and Paap (1986) have compared scaling representations derived for the same set of items using either the pairedcomparison method or the sorting method.

The effect of instructions. Concepts can be related along a variety of dimensions (e.g., apple - fire truck, bird - robin, hamburger - french fries, pen - type writer) and as a result, it is difficult to decide what instructions should be given to the judges concerning relatedness. Typically, general "relatedness" instructions have been given under the assumption that judges will choose the most relevant dimensions on which to compare the items. We have observed, however, that this lack of guidance can lead to difficulties when the data are scaled (e.g., uninterpretable dimensions using MDS) or even later when the results of the scaling analysis are applied. For example, if raters are asked to judge the relatedness of a set of editor functions that occur together in task sequences (e.g., find and change might be frequently used together) and that also have featural similarities (e.g., insert and delete might require similar command sequences), then judges may use different dimensions for different pairs of functions, rather than weighing each pair on the same set of dimensions. McDonald, Dayton, and McDonald (in preparation) found that quite different scaling representations were obtained by giving instructions to either judge the similarity of items or to judge how well items go together. In general, the relatedness instructions should be carefully considered in the data collection phase of this methodology.

#### Scaling

As part of the process of deciding how the estimates of distance are to be obtained, the scaling analysis to be used should also be considered. These two stages (obtaining distance estimates and scaling) may interact, since certain data collection methodologies emphasize (or allow) certain characteristics of the data to emerge (such as hierarchical arrangement) while other methods do not. Furthermore, once a particular scaling technique has been selected, decisions must be made concerning the particular algorithm (e.g., maximum or minimum hierarchical cluster analysis) and/or parameters (e.g., q and r for Pathfinder) for that technique.

Selection of a scaling technique. We believe that multiple scaling techniques should be used in combination to elicit knowledge. Each of the techniques emphasizes different features in the data. For instance, Pathfinder captures information about local relations (pairs of items that are highly related), whereas multidimensional scaling captures information about global relations among the item set as a whole (i.e., dimensions). Furthermore, the techniques differ in terms of the type of representation (e.g., spatial, network, hierarchical) that they generate. A particular representation or combination of representations might be desirable for a particular application. Research is needed that compares the scaling techniques in terms of psychological validity and the type of knowledge elicited. Given this information, a mapping could be derived between features of specific domains and scaling techniques. In one study that investigated this issue, Cooke, Durso, and Schvaneveldt (1986) compared the multidimensional scaling technique with the Pathfinder technique and found that Pathfinder was superior in terms of the elicitation of knowledge that was related to recall.

Selection of a particular solution. Given a particular scaling technique, the choice of parameters can greatly influence the resulting solution. There are several criteria that can be used to select a particular solution. A solution may be selected because it is an intuitively appealing representation of the domain. However, this criterion is subject to the same criticism as introspective techniques. A more objective approach is to select the solution that corresponds most closely to behavioral measures such as recall or categorization performance. The criterion that is most often used to choose among various solutions concerns "goodness of fit." The solution is selected that best fits the original data (i.e., distance estimates). Of course, the more parameters in the solution (e.g., links, dimensions) the better the fit, but because the general goal of this approach is to reduce the original set of data, it is desirable to select the solution in which fit is maximized while number of parameters is minimized. It should be noted that use of the fit criterion assumes that the original distance estimates accurately reflect knowledge. It could be that the best representation is one that least resembles the original ratings. Generally, the criteria are heuristics, not formal rules. Research is needed to make these notions more precise and to evaluate solutions resulting from varying parameters. In the Cooke, Durso, and Schvaneveldt (1986) study, a comparison was also made among various multidimensional scaling representations and various network representations of the same data. Results indicated that some representations generated by the same general technique were better at predicting recall than representations with slightly different parameters.

#### Interpretation of the Solutions

Once scaling solutions have been generated they must be mapped onto the particular application. For example, in the interface design application mapping can consist of taking the output of the scaling analysis selected and projecting it rather directly onto the organization of the interface. Even with this simple approach, however, there is ample room for interpretation. If hierarchical clustering is used, for example, there are numerous ways of emphasizing the interand intra-cluster relationships, ranging from a traditional dendrogram to a spatial arrangement of items (cf., McDonald, Stone, & Liebelt, 1983). In the knowledge elicitation application, scaling solutions need to be mapped onto a knowledgebase. This mapping can occur indirectly, by using the solutions to guide structured interviews which in turn generate information for the knowledge-base, or by transforming the solutions into a format compatible with a particular knowledge-base architecture (e.g., production rules). The scaling solutions could also be directly mapped into the knowledge-base of a system capable of interpreting it. In any case, the scaling solutions require some form of interpretation before they can be used by a knowledge-based system.

Because the interpretation process is not formalized, it is subject to numerous biases. Thus, the identification of dimensions in a multidimensional scaling analysis, a cutoff point in a hierarchical clustering solution, and link labels in a Pathfinder network are not specified by the scaling techniques, yet the interpretation of the resulting representation is highly dependent on this type of information. For example, Pathfinder networks only become semantic networks with the interpretation of links. Without link labels, caution must be exercised in interpreting and comparing links in the networks. In the knowledge engineering application, it is necessary to know not only *that* two concepts are related, but *how* they are related as well. In order to provide the Pathfinder network with semantics, the links must be interpreted or labeled. Cooke and McDonald (1986) have taken some preliminary steps towards a formal methodology for classifying and labeling links according to semantic relationships. The identification of semantic relations or dimensions is one means of providing the scaling representations with semantics. Another is through the identification of levels of abstraction in these structures. In any domain there are some concepts that are superordinate categories of other concepts or that are simply more general than other concepts. Knowledge concerning levels of abstraction enables inferences to be drawn about concepts at one level of abstraction based on concepts at a higher (i.e., more general) level of abstraction. One could envision a multi-leveled scaling representation in which each successive level contained concepts that were increasingly abstract. In the study that follows, a technique is investigated for identifying abstract concepts or categories in a set of Unix commands.

We have suggested that scaling representations can be used as a starting point for further knowledge elicitation or machine learning approaches. On the other hand, it is possible that the scaling approach could eventually result in a complete knowledge-base. If the representation is to be complete, however, a way of interpreting it is needed. (Hayes, 1985). That is, even if the details of a representation (i.e., link labels, dimensions) are identified, it is still necessary to determine how to make use of the representation. The representations themselves are not capable of making inferences, answering questions, making decisions, or solving problems. For instance, a production system could be used to reason over facts in a semantic network knowledge-base (cf. Anderson, 1983). Some preliminary work has been done on reasoning over Pathfinder networks using principles of spreading activation in a connectionist fashion (R. W. Schvaneveldt, personal communication). This approach to interpretation eliminates the need for semantic link labels.

# THE ELICITATION OF LEVELS OF ABSTRACTION

In the previous section several research issues were outlined along with some efforts toward resolving these issues. In this section we elaborate on the issue of levels of abstraction and discuss a study designed to build representations with multiple levels of abstraction.

As previously mentioned, scaling techniques require interpretation in order to apply them to many real-world situations. One potentially useful type of information not explicitly conveyed in scaling solutions concerns the differentiation among concepts according to levels of abstraction. For instance, Pathfinder networks are "flat" in the sense that relations between concepts at the same level (e.g., property relations such as bird - feathers or robin - red) and relations between concepts at different levels (e.g., superordinate relations such as bird robin or animal - bird) are not distinguished. Whereas hierarchical clustering schemes do produce multilevel structures, the levels do not necessarily correspond to actual levels of abstraction due to possible error in the data and the fact that the resulting solution needs to be completely connected. In fact, hierarchical clustering schemes may generate a multileveled structure even when the concepts are all from the same level. In order to concretize these issues and provide necessary background for the subsequent study on abstraction, results from the McDonald, Dearholt, et al. (1986) study on Unix commands is summarized below. The purpose of the McDonald, Dearholt, et al. study was to elicit knowledge about Unix commands from experienced users and to employ this knowledge in the development of an Interactive Documentation Guide for the Unix on-line documentation system. Each of 219 Unix commands was printed on an index card and 15 experienced Unix users were each asked to sort the cards based on the functions of the commands. Subjects also indicated whether or not they were familiar with the commands. A matrix of distance estimates was obtained by

counting for each pair of commands the number of subjects that placed that pair in the same pile and subtracting this value from 16. A conditional probability matrix was then created by dividing each of these co-occurrence values by the smaller of the two frequencies (number of judges familiar with each item) for that pair. This procedure tends to reduce the relative distances between pairs in which one or both items are less common. Only 44 of the 219 commands were sorted by (and thus were familiar to) all 15 experienced Unix users. Of these, 37 commands were identified as "core" Unix commands by eliminating those commands which were not grouped with one or more of the other 43 commands by at least half of the raters. While scaling analyses were performed on the data from all 219 commands, for the sake of simplicity the hierarchical cluster analysis (maximum method) of the 37 core commands is shown in Figure 3.



Figure 3. A hierarchical cluster (maximum method) analysis of 37 "core" Unix commands.

ł

In Figure 3 a cutoff value of 1 indicates that all 15 experienced Unix users grouped the commands in that cluster. The higher the cutoff, the fewer subjects who sorted the connected commands together. Even though there are 15 cutoff levels, it is unlikely that there are 15 meaningful levels of abstraction for this set of commands. Thus it becomes important to distinguish between clusters that form because of categorical associations and clusters that form because of

variability in the data. Additionally, the cluster analysis provides no information concerning the category labels for particular clusters. In fact, the nameability of each cluster might differentiate meaningful clusters from clusters that are artifacts. Clusters that are easily named are probably more meaningful than clusters that are not easily named. The purpose of the following study was to elicit information about levels of abstraction of the Unix commands from experienced Unix users in the form of "goodness of cluster" ratings and cluster names.

#### Method

Subjects. Four experienced Unix users from New Mexico State University voluntarily participated in this study. These subjects had also participated in the first Unix study.

Materials. Materials consisted of 83 clusters of two or more Unix commands taken from the original set of 219. The clusters were obtained as follows. A minimum cluster analysis was performed on the 152 commands that at least eight of the 15 experienced Unix users were familiar with. A cutoff value of ten or more in this particular solution resulted in a total of 83 clusters (some of these were nested in that a small cluster may also be included in a larger cluster). Thus, the 83 clusters contained commands that at least half of the experienced Unix users were familiar with and that belonged together, that is, at least five judges felt that the items belonged with one of the other 151 commands. Because seven of the 152 commands did not cluster in the solution at the cutoff value of ten, they did not appear at all in the 83 clusters.

**Procedure.** Each subject was seated in front of a CRT upon which each of the 83 clusters was randomly presented one at a time. For each cluster the subject was asked to respond by pressing the "0" key if there were one or more items in the cluster that were unfamiliar. If the subject responded with a "0" the next cluster was presented. Otherwise, the subject was asked to rate the "goodness" of the cluster by entering a rating of "1" through "5." A response of "1" indicated that the cluster was very bad and a response of "5" indicated that the cluster was very bad. Following the rating, subjects were asked to assign a name to the cluster. After the cluster was named, the next cluster was presented. Trials continued in this fashion until all 83 clusters had been presented.

## **Results and Discussion**

In general, the four subjects tended to agree on goodness of category ratings for the 83 clusters (mean interjudge correlation -r (81) = .97, p < .001). Ratings of "0" through "5" occurred 4.2, 9.0, 7.2, 14.5, 22.0, and 43.1 percent of the time, respectively. Thus, the distribution of the ratings was skewed with the higher ratings of "4" or "5" occurring most frequently suggesting that the cluster analysis did indeed produce mostly meaningful categories.

For each of the clusters, the "goodness" ratings were averaged over the four subjects. In addition, if the subjects agreed on a name (or a significant part of a name) for a cluster then that name was assigned to the cluster. In many cases, names tended to migrate from smaller clusters to a larger superset cluster. That is, subjects often assigned the same name to a subset of commands as they did to the larger superset. In this case the smaller subset clusters were collapsed and the name was assigned to the superset cluster. The name migration criterion, along with changes in the "goodness" rating allowed the elimination of many clusters resulting in a simpler, but more meaningful, representation. An example of this procedure for a subset of 14 of the 152 commands is presented in Figure 4. In Figure 4a the original cluster analysis (minimum method) for this subset is presented along with the average "goodness" ratings. In Figure 4b the clusters are collapsed and labeled. It should be noted that for this particular subset, name migration was responsible for all of the collapsing since all of the clusters were judged to be quite good. That is, subjects labeled *talk* and *write* as *interactive communication* as well as *talk*, *write*, and *wall* and *talk*, *write*, *wall*, and *mesg*. Thus, in this example, information about category names, reduced the number of clusters from ten to five.



Figure 4. Hierarchical cluster analysis (minimum method) with average judged "goodness" (a) and after reduction based on naming (b).

There are, however, some anomalies in the resulting representation. For instance, the command *biff* (which informs the system whether or not you want to be notified when mail arrives during the current terminal session) does not cluster in the hierarchical cluster analysis with the other four electronic mail commands until later when it also clusters with the commands related to news. We are currently working on methods for extracting information about categories from Pathfinder network solutions. A Pathfinder representation of the same subset of commands is shown in Figure 5. The three main categories are also apparent in this representation and can be identified by rich interconnections among items within the same category. We are in the process of quantifying such information so that it can be used to define category boundaries. Note that in this representation of the data, the *biff* command appears to be part of the mail commands.



Figure 5. Pathfinder solution for communication commands  $(r = \infty, q = n - 1)$ .

These results suggest that information about levels of abstraction can be obtained as a second step in the representation process. Both naming and rating appear to provide information that aids in the identification of "real" categories. The resulting representation appears to be more complete as a characterization of experienced user knowledge.

## CONCLUSIONS

We feel that the psychological scaling approach to knowledge elicitation has numerous advantages over traditional knowledge elicitation techniques, notably the formalization of the elicitation process. Results that have been obtained in other areas using this same approach have been quite promising. These techniques have been successful in the elicitation of knowledge from experts in domains such as computer programming, flight maneuvers, and the Unix operating system. However, there are several issues that should be addressed in order to refine this methodology so that it is most useful for knowledge elicitation.

A long-term goal of our research is to develop a system that embodies the knowledge elicitation tool kit along with expertise concerning the appropriate use of the tools. Thus, this knowledge elicitation expert system would be able elicit a set of domain-related items, select an appropriate data collection methodology, choose appropriate scaling techniques, carry out the scaling, select a particular scaling solution, and interpret the resulting solution. The formalization of all aspects of this methodology is needed in order to develop such a system.

On the other hand, the application of this methodology is not at a stand-still until these issues are resolved. Indeed, it has already been applied to several problems and results have been encouraging. The resolution of the issues would serve to formalize many of the aspects of this methodology that are currently left to the judgment of the experimenter or knowledge engineer. At any rate, we believe that even in its current state, this methodology constitutes a more formal and objective approach to knowledge elicitation than other existing techniques.

#### ACKNOWLEDGEMENTS

The authors would like to acknowledge Donald Dearholt, Kenneth Paap, and Roger Schvaneveldt for their valuable contributions to this paper. Thanks also to the experienced Unix users who participated in the abstraction study.

#### REFERENCES

Anderson, J. R. (1983). The Architecture of Cognition, Cambridge, MA: Harvard University Press.

Butler, K. A. & Corter, J. E. (1986). The use of psychometric tools for knowledge acquisition: A case study. In W. Gale (Ed.), *Artificial Intelligence and Statistics*, Reading, MA: Addison-Wesley.

Cooke, N. M. & Schvaneveldt, R. W. (1986). The evolution of cognitive networks with computer programming experience. Paper presented at the Workshop on Empirical Studies of Programmers, June 5-6, 1986, Washington, D.C.

Cooke, N. M., Durso, F. T., & Schvaneveldt, R. W. (in press). Recall and measures of memory organization, Journal of Experimental Psychology: Learning, Memory, and Cognition.

Cooke, N. M., & McDonald, J. E. (1986). A formal methodology for acquiring and representing expert knowledge. *IEEE Special Issue on Knowledge Representation*.

- Ericsson, K. A., & Simon, H. A. (1984). Protocol Analysis: Verbal Reports as Data, Cambridge, MA: MIT Press.
- Gammack, J. G., & Young, R. M. (1985). Psychological techniques for eliciting expert knowledge. In M. A. Bramer (Ed.), Research and Development in Expert Systems, London, UK: Cambridge University Press, 105-112.

Hayes, P. J. (1985). Some problems and non-problems in representation theory. In R. J. Brachman & H. J. Levesque (Eds.), *Readings in Knowledge Representation*, Los Altos, CA: Morgan Kaufmann, 3-22.

Hoffman, R. R. (in press). The problem of extracting the knowledge of experts from the perspective of experimental psychology. AI Magazine.

- Johnson, S. C. (1967). Hierarchical clustering schemes. *Psychometrika*, 32, 241-254.
- Kruskal, J. B. (1977). Multidimensional scaling and other methods for discovering structure. In Enslein, Ralston, & Wilf (Eds.), *Statistical Methods for Digital Computers*. New York: Wiley.

Kruskal, J. B., & Wish, M. (1978). Multidimensional Scaling. Sage University Paper series on Quantitative Applications in the Social Sciences, #07-011, London: Sage Publications.

McDonald, J. E., Dayton, J. T., & McDonald, D. R. (in preparation) Adapting menu layout to tasks.

- McDonald, J. E., Dearholt, D. W., Paap, K., & Schvaneveldt, R. W. (1986). A formal interface design methodology based on user knowledge. Proceedings of Human Factors in Computer Systems, CHI '86, 285-290.
- McDonald, J. E., Stone, J. D., Liebelt, L. S., & Karat, J. (1982). Evaluating a method for structuring the user-system interface. Proceedings of the 26th Annual Meeting of the Human Factors Society.

- McDonald, J. E., Stone, J. D., & Liebelt, L. S. (1983). Searching for items in menus: The effects of organization and type of target. Proceedings of the 27th Annual Meeting of the Human Factors Society.
- Murphy, G. L., & Medin, D. L. (1985). The role of theories in conceptual coherence. *Psychological Review*, 92(3), 289-316.
- Nisbett, R. E., & Wilson, T. D. (1977). Telling more than we can know: Verbal reports on mental processes. *Psychological Review*, 8, 231-259.
- Rips, L.J., Shoben, E. J., & Smith, E. E. (1973). Semantic distance and the verification of semantic relations. Journal of Verbal Learning and Verbal Behavior, 12, 1-20.
- Roske-Hofstrand, R. J., & Paap, K. R. (1986, November). Cognitive JNDs: Implications for Scaling Techniques. Paper presented at the 27th Annual Meeting of the Psychonomic Society, New Orleans.
- Roske-Hofstrand, R. J., & Paap, K. R. (in press). Cognitive networks as a guide to menu organization: An application in the automated cockpit. *Ergonomics*.
- Schvaneveldt, R. W., & Durso, F. T. (1981). Generalized semantic networks. Paper presented at the meetings of the Psychonomic Society.
- Schvaneveldt. R. W., Durso, F. T., & Dearholt, D. W. (1985). Pathfinder: Scaling with Network Structures. *Memorandum in Computer and Cognitive Science*, MCCS-85-9, Computing Research Laboratory, New Mexico State University.
- Schvaneveldt, R. W., Durso, F. T., Goldsmith, T. E., Breen, T. J., Cooke, N. M., Tucker, R. G., & DeMaio, J. C. (1985). Measuring the structure of expertise. International Journal of Man-Machine Studies, 23, 699-728.
- Shepard, R. N. (1962a). Analysis of proximities: Multidimensional scaling with an unknown distance function. I *Psychometrika*, 27, 125-140.
- Shepard, R. N. (1962b). Analysis of proximities: Multidimensional scaling with an unknown distance function. II. *Psychometrika*, 27, 219-246.
- Shoben, E.J. (1983). Application of multidimensional scaling in cognitive psychology. Applied Psychological Measurement, 7, 473-490.
- Tullis, T. S. (1985). Designing a menu-based interface to an operating system. Proceedings of Human Factors in Computing Systems (CHI '85) Conference.

Tversky, A. (1977). Features of similarity. Psychological Review, 84, 327-352.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# A Foundational Approach to Autonomous Knowledge Acquisition in Knowledge-Based Systems

#### James P. Delgrande

School of Computing Science. Simon Fraser University, Burnaby, B.C., Canada V5A 1S6

## ABSTRACT

A formal, foundational approach to autonomous knowledge acquisition and its relation with logical knowledge representation systems is presented. Formal systems are derived by means of which the set of potential hypotheses that can be formed on the basis of a stream facts is precisely specified. A procedure is also derived for restoring the consistency of a set of hypotheses after conflicting evidence is encountered. The framework is intended both as a basis for the development of autonomous systems that learn from examples, and as a neutral point from which such systems may be viewed and compared. The relation between acquiring knowledge within this framework and reasoning deductively is also explored.

#### 1. Introduction

Broadly speaking, there are two actions that can be performed on a body of knowledge: information may be derived from it, or the body of knowledge may be modified. In knowledge representation the foremost concern is typically to develop a sufficiently precise notation for the representation of knowledge, together perhaps with an inferential apparatus for reasoning with the knowledge. In knowledge acquisition the concern is to develop methodologies and techniques for incorporating domain knowledge into representation structures. This paper presents a formal, foundational approach to the acquisition of knowledge. The basic idea is to ignore, insofar as possible, representational concerns and instead develop in isolation (again, insofar as possible) a framework for autonomous knowledge acquisition. To this end the starting point is the same as that of virtually all knowledge representation schemes: that the world consists of, or may be described by, a collection of individuals and a collection of relations on these individuals [Mylopoulos and Levesque 84]. Such a framework for knowledge acquisition could have two potential benefits: first it could provide a formal standpoint from which acquisition systems may be constructed and from which existing acquisition systems may be examined and compared. Second it could provide a standpoint from which the concerns of acquisition, and how they impinge on concerns of deductive reasoning with a knowledge base, may be examined.

These notions are expanded and amplified in the rest of this section. In the following section, formal systems are developed for characterising this approach to knowledge acquisition, and the approach is compared with representative AI learning systems. Following this, the relationship between acquisition systems which fall into this framework, and deductive reasoning systems, is examined. Further details, proofs of theorems, etc. are given in [Delgrande 85].

#### 1.1. Approach

The world is assumed to be describable by a collection of individuals and relations on individuals. This, of course, is the starting point for the standard Tarskian definition of an interpretation for (function-free) first-order logic. Clearly any learning system should be able to receive information or "facts" concerning this world or domain, and so at any point in time, some finite portion of the domain, described by a set of ground atomic formulae, is assumed to be known by the learning system. Moreover, as time progresses, a learning system will encounter new information, and so this set of known ground atomic formulae may monotonically increase. Also it is initially assumed that the only information available to a learning system concerning the domain is in the form of ground instances. The reason for this is that if more general statements were permitted - for example existentials. universal generalisations, or disjunctions - their presence would beg the question of their origin. Thus, for example, if we were told that there was a white raven in Ontario, this would presuppose some agent who has knowledge (somehow) of this individual. Note though that if we knew that a particular individual was a raven and white we would also know that there exists a white raven, but only as a consequence of the ground instances. Similarly, if we were informed that elephants had four legs, this would presuppose an agent who had determined this relation. In any case, such information presupposes an entity that had already performed some acquisition, or who had formed more complex sentences from the ground data. So, to begin with, I assume that the only available information is in the form of ground formulae. However, this assumption is restrictive<sup>1</sup>, in that it excludes more general statements, and so, after appropriate systems have been developed, consequences of relaxing this assumption are investigated.

Given a set of ground instances, it seems reasonable to propose a structure for this set, and hence hypothesise relations among the known relations. Thus if all ravens we had encountered were black, we might form the hypothesis "all ravens are black". However, since the set of known ground instances increases monotonically, new hypotheses may become tenable while others will be falsified. So the question arises as to how a set of hypotheses may be modified as falsifying instances are encountered.

This part of the problem then consists of forming hypotheses based on a stream of ground atomic formulae, where the hypotheses are phrased in terms of set relations among (the extensions of) predicates. However the emphasis here is on what hypotheses may *potentially* be formed, and not on which hypotheses may *justifiably* be formed. Thus for example if we have a set of black ravens and know of no non-black ravens, we could hypothesise that ravens are black. However the approach at hand gives no indication as to when such a hypothesis should be formed or what constitutes adequate evidence for such an assertion. So the problem is to determine formal criteria which prescribe the set of potential conjectures, rather than to determine pragmatic criteria whereby an acceptable set of conjectures may be formed. A similar distinction can be made in a deductive system, where an underlying logic specifies what *could* be derived, but not what *should* be derived.

The overall approach then is as follows. Hypotheses are proposed and modified on the basis of a finite, monotonically increasing set of ground instances. The hypotheses are expressed in a language, HL, that is a simple variant of the language of elementary algebra. The criteria for proposing hypotheses are straightforward: there is a reason to do so (i.e. some minimal notion of evidence is satisfied) and the hypothesis is not known to be falsified. These criteria though are far too simplistic, and in general the resultant set of hypotheses

<sup>&</sup>lt;sup>1</sup> But not so restrictive as it may at first seem. As mentioned, an assignment of individuals and relations is all that is assumed in a Tarskian interpretation. Moreover, there are significant applications which make similar restrictive assumptions. Relational databases clearly consist of ground formulae, and sentences of, for example, relational algebra are equivalent to such sets. Many semantic networks expressly limit or omit disjunctive and existential sentences, offsetting the loss of expressibility with a gain in computational efficiency.

will be inconsistent.

However, with each term in a sentence of HL we can associate two subsets of the ground instances, consisting of those known to satisfy the term and those known to not. For example, to the term "black raven" we can associate the set of individuals known to be black and a raven, and the set known to be either non-black or non-raven. Formal systems are developed to characterise relations between terms in HL by means of these sets. From this, ground instances whose truth values are unknown can be iteratively located so that determining their truth value leads to a convergence of the hypothesis set to consistency. Informally these "knowable but unknown ground instances" correspond to unknown but potentially knowable "facts" in the domain.

This process provides a sharp separation between the inductive and deductive aspects of the problem of acquisition. Induction, as such, plays a relatively minor role: it is used to suggest an initial (and usually inconsistent) set of hypotheses, which then are modified using strictly deductive techniques. The set of hypotheses that may be formed is shown to be perhaps surprisingly general and in fact (with respect to expressiveness) subsumes a number of systems for learning from examples and by discovery.

To recapitulate then, for knowledge acquisition I assume only that:

- 1. the domain is describable as a set of ground atomic formulae, some finite subset of which is known;
- 2. the set of known ground atomic formulae is correct and error-free;
- 3. the set of known ground atomic formulae may grow monotonically with time; and
- 4. known individuals or tuples of known individuals may be tested for membership in a known relation.

Of these assumptions, the second is clearly the most restrictive, and deserves further comment. Consider where we have some conjecture (say, "ravens are black") and an exceptional individual (say, we encounter an albino). If we don't want to totally abandon our original hypothesis, then there seems to be two ways we can discharge the exception. First, we could amend the conjecture to something like "normally ravens are black", and perhaps also introduce "normally albino ravens are white". Formal aspects of this approach are investigated in [Delgrande 86] — in any case the exception is "excused". Second we could determine, or simply declare, that the observation is erroneous: either the individual is not a raven or it really is black. However, *this* procedure of determining that an observation is incorrect is a pragmatic concern, and is quite distinct from our concern of what hypotheses "follow" potentially from a set of observations.

Once such a framework for knowledge acquisition is established, we can examine the relationship between such acquisition systems and knowledge representation schemes. Since acquiring knowledge of a domain and deductively reasoning with this knowledge are two fundamentally different enterprises, the resulting systems may be expected to differ significantly. This in fact proves to be the case. Given our assumptions, the natural direction for investigating acquisition from a stream of examples leads to a set-theoretic approach, while the natural direction for reasoning about properties and relations among individuals leads to a (first-order) logical approach. This part of the problem also addresses the issue of how arbitrary statements, in addition to the ground instances, can be incorporated into the acquisition process.

#### 1.2. Related Work

Much work has, of course, been carried out in AI addressing the problem of learning from a stream of examples. The early, influential work of Patrick Winston is described in [Winston 75]; in this system, descriptions of concepts are formed from a set of examples of the concept and "near misses". The work presented in [Brown 73], [Hayes-Roth 78], [Mitchell 77], [Shapiro 81], [Solway and Riseman 77], and [Vere 78] also falls into this category. The

more recent work of Michalski, presented in [Michalski 83], is a particularly detailed approach to learning from examples. An extensive survey of AI learning systems is given in [Dietterich et el 82], while [Smith et al 77] describes a proposed "model" for learning from examples, and [Dietterich and Michalski 83] compares four particular generalisation programs. Much of this work is concerned with proposing and refining a description of a concept. In contrast, the work at hand deals with characterising the hypotheses formable under a set of (arguably) minimal assumptions and hence is more concerned with exploring intrinsic properties and limitations of such approaches. In the last subsection of the next section I return to this distinction and compare the work at hand with three systems for learning from examples.

Most formal approaches to learning from examples have been concerned with inducing instances of a given type of formal language. The area of *learning theory* studies systems that implement functions from evidential states to languages. Learning theory was given a rigorous foundation with the work of Gold [Gold 67]. A survey of such approaches is presented in [Angluin and Smith 82], while [Osherson et el 83] gives recent results in this area. The key difference between such approaches and the present work, clearly, is that no underlying formal grammar is presupposed, beyond that for elementary set theory.

#### 2. Introducing Conjectural Information

#### 2.1. A Language for Introducing Conjectures

As mentioned, the domain of application is assumed to be described by a presumably infinite set of ground atomic formulae, formed from presumably infinite sets of individuals and predicates. However, given a particular predicate, all that can be known of it is a subset of those individuals (or tuples) which satisfy it and a subset of those individuals which do not. I will speak of an individual as being *known*, if it is known to be or not be part of the extension of a known predicate. A predicate will be referred to as *known* if its truth value on a given individual (tuple) can be determined<sup>2</sup>. Informally, a known individual or predicate is one "encountered" by a learning system. The sets of tuples known to belong to the extension of a predicate and known to not belong to the extension are referred to as the *known extension* and the *known antiextension* respectively. So for a known *n*-place predicate *P* and known individuals  $a_1, \ldots, a_n$  there are three possibilities:

1.  $P(a_1, \ldots, a_n)$  is known to be true.

2.  $\neg P(a_1, \ldots, a_n)$  is known to be true.

3. Neither  $P(a_1, \ldots, a_n)$  nor  $\neg P(a_1, \ldots, a_n)$  are known to be true.

**Definition:** For each known predicate symbol P define sets  $P_+$  and  $P_-$  by:

$$P_+ = \{\langle a_1, \ldots, a_n \rangle \mid P(a_1, \ldots, a_n) \text{ is known to be true} \}.$$

$$P_{-} = \{ \langle a_1, \ldots, a_n \rangle \mid \neg P(a_1, \ldots, a_n) \text{ is known to be true} \}.$$

Conjectures are expressed in a language HL. This language is analogous to that of elementary set theory, except that operators and relations are subscripted with the character "h". I will use the symbol " $|_h$ " for the (hypothesised) disjointness relation, and " $\alpha_h$ ", " $\circ_h$ ", and " $\uparrow_h$ " for the converse, composition, and image operations. A "ply" operator  $\supset_h$  is also introduced. When we come to consider the algebra of terms of HL, we will also want to consider the corresponding propositional logic. The ply operator will serve as the analogue in the algebra of the material conditional in the logic. This operator is discussed further later in this section.

<sup>&</sup>lt;sup>2</sup> Thus perhaps, for a known predicate, a verification procedure is assumed to be known for determining if a given individual is part of the extension.

Definition: If P is the set of known predicate names, then the *terms* of HL are exactly those given by:

- 1) If  $\alpha \in \mathbf{P}$  then  $\alpha$  is a term of HL.
- 2) If  $\alpha$ ,  $\beta$  are 2-place terms and  $\gamma$  a 1-place term of HL not containing  $\supset_h$ , then  $\alpha_h \alpha$ ,  $\alpha_{\circ h} \beta$ , and  $\alpha \hat{1}_h \gamma$  are terms of HL.
- 3) If  $\alpha$  and  $\beta$  are terms of HL, then so are  $\alpha \cap_h \beta$ ,  $\alpha \cup_h \beta$ ,  $\neg_h \alpha$ , and  $\alpha \supset_h \beta$ .

Definition: The sentences of HL are exactly given by:

If  $\alpha$ ,  $\beta$  are terms of HL, then  $\alpha =_h \beta$ ,  $\alpha \subset_h \beta$ ,  $\alpha \subseteq_h \beta$ ,  $\alpha |_h \beta$ ,  $\alpha \neq \beta$ ,  $\alpha \not\subset \beta$ ,  $\alpha \not\subseteq \beta$ ,  $\alpha \nmid \beta \in HL$ .

So, for example,

 $Raven \cup_h Penguin \cup_h Robin \subseteq_h Bird$ 

 $Uncle =_{h} (Brother_{h}Parent) \cup_{h} (Husband_{h}Sister_{h}Parent)$ 

have the respective readings "the set of ravens, penguins, and robins is hypothesised to be contained in the set of birds" and "(the binary relation) uncle is hypothesised to be equivalent to the union of the composition of brother with parent, and the composition of husband with sister with parent".

The known extension and antiextension corresponding to terms in HL can easily be determined. Thus, for example, the hypothetical intersection of P and Q is known to contain just those elements that both P and Q are true of, and is known to not contain just those elements that either P or Q is known to not be true of. For the hypothesised operations we obtain:

Proposition:

Complement:	$\neg_h P = (P, P_+)$
Union:	$P\cup_{h}Q = (P_{+}\cup Q_{+}, P_{-}\cap Q_{-})$
Intersection:	$P \cap_h Q = (P_+ \cap Q_+, P \cup Q)$
Ply:	$P \supset_h Q = ((\neg P_+ \cup Q_+) \cap (\neg Q \cup P), Q \cap \neg P)^3$
Converse:	$\alpha_h P = (\{ \langle y, x \rangle   \langle x, y \rangle \in P_+ \}, \{ \langle y, x \rangle   \langle x, y \rangle \in P \})$
Image:	$P \uparrow_h Q = (\{y \mid (\exists x) (\langle x, y \rangle \in P_+ \land x \in Q_+)\}, \emptyset)$ for binary relation P and one-place predicate Q
Composition:	$P_{\bullet_h}Q = (\{\langle x,z \rangle   (\exists y)(\langle x,y \rangle \in P_+ \land \langle y,z \rangle \in Q_+)\}, \emptyset).$

These operations generalise easily to ternary and higher-order predicates. Other operations such as domain and range may be defined in terms of these.

It will prove essential that occasionally we be able to determine the value of  $P(a_1, \ldots, a_n)$ , provided that P and  $a_1, \ldots, a_n$  are known. However, in general we would not want to determine the truth values of all known predicates applied to all combinations of known individuals. The reason for this is combinatorial: given p known n-place predicates and m individuals, there are  $pm^n$  knowable ground instances. In the approach to be described, at most p(p-1) of these combinations need to be known for hypothesising relations.

Naively, two terms of HL may be conjectured to be equal when there is some reason to do so (i.e. the intersection of their known extensions is non-empty) and there are no known counter-examples. While there are conditions other than these that can be used for forming conjectures, any set of alternative conditions arguably must include at least these, and so these conditions represent a set of minimal criteria for hypothesis formation. (See [Del-grande 85] for an examination of other such conditions.) In a similar manner, conditions for containment and disjointness may also be specified.

 $<sup>^{3}</sup>$  This expression is chosen so that it corresponds to the material conditional in the logic (following).

We obtain:

Definition:

 $\alpha = {}_{h}\beta \text{ when } \alpha_{+} \cap \beta_{+} \neq \emptyset \text{ and } \alpha_{+} \cap \beta_{-} = \emptyset \text{ and } \alpha_{-} \cap \beta_{+} = \emptyset.$   $\alpha \subset_{h}\beta \text{ when } \alpha_{+} \cap \beta_{+} \neq \emptyset \text{ and } \alpha_{+} \cap \beta_{-} = \emptyset \text{ and } \alpha_{-} \cap \beta_{+} \neq \emptyset.$   $\alpha \subseteq_{h}\beta \text{ when } \alpha_{+} \cap \beta_{+} \neq \emptyset \text{ and } \alpha_{+} \cap \beta_{-} = \emptyset.$  $\alpha |_{h}\beta \text{ when } \alpha_{+} \cap \beta_{+} = \emptyset \text{ and } \alpha_{+} \cap \beta_{-} \neq \emptyset \text{ and } \alpha_{-} \cap \beta_{+} \neq \emptyset.$ 

The problem with this approach to forming conjectures, of course, is that it is hopelessly simplistic. For example, assume that  $MSc\_Sup$  means that one can supervise M.Sc. students, while  $PhD\_Sup$  and  $Has\_PhD$  means that one can supervise Ph.D. students or has a Ph.D. (respectively). If all that is known is that

MSc\_Sup(John), PhD\_Sup(John), Has\_PhD(John), together with MSc\_Sup(Mary), ¬PhD\_Sup(Mary)

we would have:

 $PhD\_Sup \subset_h MSc\_Sup, PhD\_Sup =_h Has\_PhD$ , along with  $Has\_PhD =_h MSc\_Sup$ .

This clearly is inconsistent. A potential solution to this difficulty is to determine the truth values of select ground instances, where both the predicate and the individual are known, but where the truth value of the ground instance is not known. In the above example, if  $Has\_PhD(Mary)$  was determined to be true, then  $PhD\_Sup \subset_h Has\_PhD$  could be formed; if  $Has\_PhD(Mary)$  was determined to be false, then  $Has\_PhD =_h MSc\_Sup$  could be weakened to  $Has\_PhD \subset_h MSc\_Sup$ .

So two questions arise. The first concerns how such "select" ground instances can be determined for the restoration of consistency. The second concerns specifying or characterising the conjectures to which this procedure may be applied. Both questions are answered by examining the algebra of the known extensions and antiextensions of terms of HL and, from this, the corresponding propositional logic.

2.2. An Algebra and a Logic for Forming Conjectures

Two terms of HL are defined to be (strictly) equal when their known extensions and antiextensions coincide. Containment ( $\leq$ ) is introduced by the usual definition. Hence:

Definition: For  $\alpha$ ,  $\beta$  terms of HL.

 $\alpha = \beta \text{ iff } \alpha_+ = \beta_+ \text{ and } \alpha_- = \beta_-.$  $\alpha \leq \beta \text{ iff } \alpha \cap_h \beta = \alpha.$  $\alpha < \beta \text{ iff } \alpha \leq \beta \text{ but } \alpha \neq \beta.$ 

The resultant algebra HLA is given by HLA = [H;  $\neg_h$ ,  $\bigcap_h$ ,  $\bigcup_h$ ,  $\supset_h$ ], where the carrier H is given by:

 $\mathbf{H} = \{ \langle a, b \rangle \mid a, b \subseteq \mathbf{I} \text{ and } a \cap b = \emptyset \}$ 

for a set of known individuals I. The pair of elements in a member of H corresponds to a possible known extension/antiextension pair. Upper and lower bounds of H are defined by:

 $\mathbf{1} =_{df} (\mathbf{I}, \emptyset) \quad \mathbf{0} =_{df} (\emptyset, \mathbf{I})$ 

We obtain the following postulates:

Postulates:

- P1  $\alpha \cap_h \beta = \beta \cap_h \alpha$   $\alpha \cup_h \beta = \beta \cup_h \alpha$ P2  $\alpha \cap_h (\beta \cap_h \gamma) = (\alpha \cap_h \beta) \cap_h \gamma$
- $\alpha \cup_{h} (\beta \cup_{h} \gamma) = (\alpha \cup_{h} \beta) \cup_{h} \gamma$
- P3  $\alpha \cap_h (\alpha \cup_h \beta) = \alpha$  $\alpha \cup_h (\alpha \cap_h \beta) = \alpha$
- P4  $\alpha \cap_h (\beta \cup_h \gamma) = (\alpha \cap_h \beta) \cup_h (\alpha \cap_h \gamma)$  $\alpha \cup_h (\beta \cap_h \gamma) = (\alpha \cup_h \beta) \cap_h (\alpha \cup_h \gamma)$
- P5  $\alpha \cap_h \alpha = \alpha$  $\alpha \cup_h \alpha = \alpha$
- P6  $\alpha \cap_h (\beta \cup_h (\alpha \cap_h \gamma)) = (\alpha \cap_h \beta) \cup_h (\alpha \cap_h \gamma)$  $\alpha \cup_h (\beta \cap_h (\alpha \cup_h \gamma)) = (\alpha \cup_h \beta) \cap_h (\alpha \cup_h \gamma)$
- P7  $\alpha \cap_h 0 = 0$   $\alpha \cup_h 0 = \alpha$  $\alpha \cap_h 1 = \alpha$   $\alpha \cup_h 1 = 1$
- P8  $\alpha = \neg_h \neg_h \alpha$
- P9  $\neg_h(\alpha \cup_h \beta) = \neg_h \alpha \cap_h \neg_h \beta$  $\neg_h(\alpha \cap_h \beta) = \neg_h \alpha \cup_h \neg_h \beta$
- P10  $\alpha \cap_h \neg_h \alpha \leq \beta \cup_h \neg_h \beta$
- P11  $\alpha \cap_h (\alpha \supset_h \beta) \leq \beta$
- R1 If  $\alpha \cap_h \gamma \leq \beta$  then  $\gamma \leq (\alpha \supset_h \beta)$

P1-P10 then characterise  $\bigcap_h$ ,  $\bigcup_h$ , and  $\neg_h$ . These postulates very nearly, but don't quite, characterise Boolean algebras. Instead of a postulate for a universal complement,

 $\alpha \cap_h \neg_h \alpha = \mathbf{0}, \qquad \alpha \cup_h \neg_h \alpha = \mathbf{1}$ 

we obtain the weaker "Kleene" postulate P10. However we retain postulates governing universal bounds (P7) and involution (P8) as well as De Morgan's laws (P9). The weakened complement arises from the fact that the known extension and antiextension of a predicate typically do not together constitute the set of known individuals I. This algebra has been investigated under the names of *normal involution lattices* [Kalman 58] and *Kleene algebras* [Kleene 52].

We also want to derive the propositional logic corresponding to HLA. The operations of hypothesised intersection, union, and complement will clearly be analogous to the logical operations of conjunction, disjunction, and negation. Corresponding to the material conditional in the logic we need to introduce a fourth operation, called the *ply* [Curry 63]. Postulates P11 and R1 are used to characterise this operation. P11 corresponds to modus ponens, while R1 says that the ply is maximal among solutions to P11. Note though that  $\supset_h$  is of limited usefulness in forming conjectures: while it does in fact correspond to the material conditional in the logic (following), logical entailment corresponds to inclusion ( $\leq$ ) in the algebra. Thus the ply serves basically to facilitate development of the formal results.

Given these postulates, the corresponding propositional logic HLL is derived. This logic, which seems to have not appeared in the literature, is given below.

#### Axiom Schemata

A1  $\alpha \supset (\beta \supset \alpha)$ A2  $(\alpha \supset (\beta \supset \gamma)) \supset ((\alpha \supset \beta) \supset (\alpha \supset \gamma))$ A3  $\alpha \land \beta \supset \alpha$ A4  $\alpha \land \beta \supset \beta$ A5  $\alpha \supset (\beta \supset (\alpha \land \beta))$ A6  $\alpha \supset (\alpha \lor \beta)$ A7  $\beta \supset (\alpha \lor \beta)$ A8  $(\alpha \supset \gamma) \supset ((\beta \supset \gamma) \supset (\alpha \lor \beta \supset \gamma))$ A9  $\alpha \equiv \neg \neg \alpha$ A10  $(\alpha \supset \neg \alpha) \lor \neg (\alpha \supset \neg \alpha)$ Rules of Inference

MP From  $\vdash \alpha$  and  $\vdash \alpha \supset \beta$  infer  $\vdash \beta$ HN  $\vdash \alpha \supset \beta$  iff  $\vdash (\alpha \supset \neg \alpha) \lor \beta$  and  $\vdash (\neg \beta \supset \beta) \lor \neg \alpha$ 

A semantic account for the formulae of HLL follows easily from the algebra HLA.

#### Definition:

 $\models \alpha'$  in HLL iff  $\alpha = 1$  in HLA.

where  $\alpha'$  is the formula obtained from  $\alpha$  by replacing intersection with conjunction, etc. in the obvious way. The material conditional is linked to containment, via entailment, by the following.

**Proposition:**  $\models \alpha' \supset \beta'$  in HLL iff  $\alpha \leq \beta$  in HLA.

We obtain:

Theorem: HLL is sound and complete with respect to HLA.

Corollary: HLL is decidable.

Not surprisingly, negation in HLL is weaker than in PC: we lose *reductio ad absurdum* as a method of proof; also we lose the law of the excluded middle. PC is obtained if A10 and HN are replaced by  $(\alpha \supset \beta) \supset ((\alpha \supset \neg \beta) \supset \neg \alpha)$ . We also obtain:

Theorem:

- 1.  $\vdash \alpha \supset \beta$  iff  $\vdash \neg \beta \supset \neg \alpha$ .
- 2. If  $\vdash \neg \alpha \lor \beta$  then  $\vdash \alpha \supset \beta$
- 3. If  $\vdash \beta$  then  $\vdash \neg \beta \supset \alpha$
- 4. If  $\vdash \beta$  and  $\vdash \alpha \supset \neg \beta$  then  $\vdash \neg \alpha$
- 5. If  $\vdash \neg(\alpha \supset \beta)$  then  $\vdash \neg\beta$

Note however that none of the formulae obtained by replacing the meta-theoretic "if  $\cdots$  then  $\cdots$ " in the above, with the material conditional, is a theorem of HLL.

The logic resembles the system of first degree entailment,  $E_{fde}$ , of [Anderson and Belnap 75]. The principal difference is that axiom A1 and the theorem  $\alpha \supset (\beta \supset \beta)$  of HLL is rejected by Anderson and Belnap, while their axiom  $(\alpha \supset \beta) \supset (\neg \beta \supset \neg \alpha)$  is not a theorem of HLL.

#### 2.3. Restoring the Consistency of Conjectures

For restoring the consistency of a set of conjectures, we subsequently obtain:

**Theorem:** If  $\alpha_1, \ldots, \alpha_n, \alpha \in HL$  and  $\alpha_1, \ldots, \alpha_n$  have been hypothesised according to our naive criteria, where  $\alpha$  is derivable from  $\alpha_1, \ldots, \alpha_n$  in HLA, then ground instances can be determined, given a set of confirming instances for  $\alpha_1, \ldots, \alpha_n$  that will either

(1-7

- 1. refute one of  $\alpha_1, \ldots, \alpha_n$
- 2. allow  $\alpha$  to be hypothesised or, if  $\alpha$  is a negation of a relation, known.
- Outline of Proof: Equality is characterised in HLA in the standard fashion as a reflexive predicate which obeys the principle of substitution of equals into formulae. An equivalent, but much more basic, characterisation is provided and shown to be equivalent to the original. This alternative set of rules then may be used for forming proofs in HLA. However, this set of rules has the important property that if there is evidence (confirming ground instances) for the rule premiss(es), then there is (immediately and simply) evidence for the conclusion, or else there is an instance that falsifies the rule premiss. Since a proof is a sequence of steps from original premisses to desired conclusion according to the rules of inference, the theorem follows immediately. Note that it does not matter which proof of  $\alpha$  from  $\alpha_1, \ldots, \alpha_n$  is selected.

This guarantees that if our naive criteria allowed us to hypothesise that

 $A =_h B, B =_h C, C =_h D,$ 

but not

A = D,

that we could identify an instance g with the following characteristics. The predicate symbol of g is among the set of known predicates, and the constants (individuals) of g are among the known individuals. The truth value of g is unknown, but determining the truth value of g will refute one of the premisses, or, via our naive criteria, allow the conclusion  $A=_{h}D$  to be hypothesised.

The proof of this theorem is constructive, and leads immediately to a procedure which will locate evidence (i.e. ground instances) for a conjecture  $\alpha$  that follows from a set of premisses  $\alpha_1, \ldots, \alpha_n$ , where each premiss has supporting evidence, or else will refute one of  $\alpha_1, \ldots, \alpha_n$ . This evidence is located from the instances supporting  $\alpha_1, \ldots, \alpha_n$ ; the procedure is linear in the length of the proof of  $\alpha$ . A corollary to the theorem allows for a generalisation of the notion of evidence to include any metric that is applied uniformly to the conjectures. Consistency can thus be restored in a set of conjectures by repeatedly applying this procedure. Moreover, if we begin solely with a set of ground instances we will, by repeated application of our naive criteria for forming conjectures together with this procedure, arrive at a set of consistent conjectures.

This resolves the first question posed above concerning how consistency can be maintained. The second is answered via the logic HLL and a secondary result concerning the algebra: the conjectures to which the procedure may be applied correspond precisely to the sentences of elementary set theory, except that

- 1. we do not have a universal complement.
- 2. the ply operator does not appear within the scope of a hypothetical converse, image, or composition operator.<sup>4</sup>

The first condition is unavoidable in this approach (or any approach that generalises solely from a stream of ground instances). For the complement, though, we do retain involution, De Morgan's laws, laws concerning universal bounds, and the "Kleene" postulate. The second condition, which was given in the definition of HL, presents no real obstacle. Since containment provides an analogue for entailment, the ply is of limited use in forming conjectures.

<sup>&</sup>lt;sup>4</sup> This condition formed part of the definition of the language HL and is required in the proof of the above theorem.

#### 2.4. Further Issues

Up to this point I have been considering the question of what conjectures may (potentially) be formed on the basis of a stream of ground instances, and how the consistency of a set of conjectures may be restored in the face of conflicting ground instances. In this section I further consider formal properties of the systems involved, along with the overall expressiveness of these systems. The approach is to consider elements of the algebra HLA (i.e. known extension and antiextension pairs) and compare them with standard (Boolean) sets.

In HLA we have entities such as  $(Red_+, Red_-)$  where  $Red_+$  is the set of things known to be (say) red and  $Red_-$  is the set of things known not to be red. These entities represent what is known of predicates (relations) in the domain of application. In standard naive set theory we simply have some (finite, known) collection of entities. So, for example, in this latter case, we may wish to distinguish some set of known entities  $\{s_1, \ldots, s_k\}$ . If we use capitalised strings for the former entities and strings beginning with a lower case letter for the latter, we might say something like:

$$Red = (Red_+, Red_-) = (\{r_1, \ldots, r_n\}, \{t_1, \ldots, t_m\})$$
 and  $s = \{s_1, \ldots, s_k\}.$ 

So entities such as *Red* must be distinguished from sets such as s. The latter are wholly and entirely known, in that their constituents are completely known. Thus  $s = \{s_1, \ldots, s_n\}$ exactly determines s; for any individual a, it is known whether  $a \in s$  or  $a \notin s$ . *Red* on the other hand is used to represent what is known about a relation in the domain of application — that is, the individuals known to be in the extension and the individuals known to not be in the extension. However there are further differences between these two types of entities.

First, if  $s = \{s_1, \ldots, s_k\}$  then any occurrence of s can always be replaced by  $\{s_1, \ldots, s_n\}$ . Thus it seems reasonable to assert Stack(s) or  $Stack(\{s_1, \ldots, s_n\})$  with equal facility. This though is not the case with *Red*. While it seems reasonable to say, for example,  $colour\_type(Red)$ , we certainly don't want to say  $colour\_type((Red_+, Red_-)))$ . Even if we knew that  $\{r_1, \cdots\}$  was the extension of *Red*, we wouldn't want to say  $colour\_type(\{r_1, \cdots\})$ . The difference lies in the intensional nature of predicates such as *Red*, an aspect not shared by sets such as s. That is, *Red*, and in fact all the elements of P and all the terms of HL, are assumed to represent what is known about properties; the conjectures of HL then may be looked on as representing hypothesised interrelations among properties, based on what is known of their extensions. This means that, among other things, we can't distinguish predicates with differing intensions but with the same extensions (for example, vertebrates with hearts from vertebrates with kidneys).

But sets such as s, which are formed from members of I and P, are defined by their extensions. Once again, if  $s = \{s_1, \ldots, s_n\}$  there is nothing else to learn about the extension of s. These sets that can always be replaced by their extension I will refer to as *reducible*. An equally appropriate term is *knowable*. The (hypothetical) sets, such as *Red*, which cannot be replaced by their extension, I will refer to as *irreducible*. An equally appropriate term here is *unknowable*.

So there are two questions of interest:

- 1) How can we formally characterise the irreducible sets?
- 2) How do the reducible and irreducible sets interrelate?

These questions are addressed by specifying, via a list of axioms, the set of allowable reducible and irreducible sets. For both reducible and irreducible sets, the axioms developed will parallel those in the system of Zermelo-Fraenkel (ZF) [Fraenkel et al 73]. Note however that there is no commitment made as to which set theory applies in the real world. That is, there are sets (maybe classes) in the real world that the irreducible sets correspond to; however there is no need to decide which theory governs those (unknowable) sets.

Omitting the development, we obtain:

#### Notation:

a, b, c,  $\cdots$  - reducible A, B, C,  $\cdots$  - irreducible  $\cdots$ , x, y, z - either reducible or irreducible

# Set Axioms:

Existence:

i) If  $a \in I \cup P$ ,  $\{a\}$  is a set.

ii) If  $a, b \subseteq I \cup P$  and  $a \cap b \neq \emptyset$  then (a, b) is an irreducible set.

Extensionality:

i)  $(a)(b)(x)(x \in a \equiv x \in b) \supset a = b$ 

ii)  $(A)(B)(x)((x \in A_+ \equiv x \in B_+) \land (x \in A_- \equiv x \in B_-)) \supset A = B$ 

Pairing:

$$(x)(y)(\exists a)(z)(z \in a \equiv (z=x \lor z=y))$$

Sum:

i) 
$$(a)(\exists b)(x)(x \in b \equiv (\exists c)(x \in c \land c \in a))$$
  
ii)  $(a)((\exists D)(D \in a) \supset$   
 $(\exists B)(x)[x \in B_+ \equiv ((\exists c)(x \in c \land c \in a) \lor (\exists C)(x \in C_+ \land C \in a))]$   
 $\land [x \in B_- \equiv \neg (\exists c) ((x \in c \land c \in a) \land (C) (x \in C_- \land C \in a))])$   
iii)  $(A)(\exists C)(x) ([x \in C_+ \equiv (\exists B (x \in B_+ \land B \in A_+) \lor (\exists b)(x \in b \land b \in A_+))]$   
 $\land C_- = \emptyset)$ 

:

Power Set:

i) 
$$(a)(\exists b)(c)(c \in b \equiv c \subseteq a)$$
  
ii)  $(A)(\exists B)(c)((c \in B_+ \equiv c \subseteq A_+) \land (c \in B_- \equiv (c \subseteq (A_+ \cup A_-) \land c \not\subseteq A_+)))$ 

Separation:

i) 
$$(a)(\exists b)(x)(x \in b \equiv x \in a \land \sigma(x))$$
 for b not free in  $\sigma$ , and  $\sigma$  reducible or irreducible.

ii) 
$$(A)(\exists B)(x)((x \in B_+ \equiv (x \in A_+ \land \sigma(x))) \land (x \in B_- \equiv (x \in A_- \lor \neg \sigma(x))))$$
  
for B not free in  $\sigma$ , and  $\sigma$  reducible or irreducible.

Regularity:

$$(a)(a \neq \emptyset \supset (\exists x)(x \in a \land (y)(y \in x \supset y \notin a)))$$
  
where for x irreducible,  $y \in x$  means  $y \in x_+$ .

This axiomatisation then answers the preceding two questions regarding reducible and irreducible sets: as well it provides us with the following benefits. First, the hypothetical operators can be justified and defined in terms of the axioms for irreducible sets. So for example, for intersection we have:

Theorem:  $(A)(B)(\exists C)(x)((x \in C_+ \equiv x \in A_+ \land x \in B_+) \land (x \in C_- \equiv x \in A_- \lor x \in B_-))$ and C is unique.

So for any two irreducible sets there is a third unique set whose known extension consists of elements common to the known extensions of the first two sets and whose known antiextension consists of elements in either of the known antiextensions of the two sets. This in turn justifies the definition:

**Definition:**  $A \cap_h B = C$  iff  $(x)((x \in C_+ \equiv x \in A_+ \land x \in B_+) \land (x \in C_- \equiv x \in A_- \lor x \in B_-)).$ 

Thus,  $A \cap_h B = (A_+ \cap B_+, A_- \cup B_-)$ . Moreover results of applying an operator to a reducible and an irreducible set can be justified and defined. For example, for intersection we obtain:

#### **Definition:** $A \cap_{b} b = c$ iff $(x)(x \in c \equiv x \in b \land x \in A_{+})$ .

Thus  $A \cap_h b = A_+ \cap b$ . Thus the result of intersecting a known set of blocks with the (irreducible) predicate *Red* is the subset of the blocks known to be red. The other hypothetical operations can be similarly defined.

Given the extended system, the concept of transitive closure is introduced in [Delgrande 85]. Given transitive closure, we can introduce a rudimentary means of relating predicates which apply to sets of objects and to predicates that apply to pairs (or some fixed number) of objects. Thus we can express the hypotheses that a stack of objects is a set of objects that satisfies the transitive closure of the On relation and, conversely, any set of objects so bounded is hypothesised to be a stack. As an example of the expressiveness that we obtain, consider the oft-cited example of an arch. In first order logic we may write:

$$(y_1)(y_2)(y_3)[Arch\_reln(y_1, y_2, y_3)] \equiv$$

$$On(y_3, y_1) \land On(y_3, y_2) \land \neg Touching(y_1, y_2) \land Pillar(y_1) \land Pillar(y_2) \land Block(y_3)$$
]

to express the fact that an arch relation obtains between  $y_1$ ,  $y_2$ , and  $y_3$ . In HL we can express the hypotheses:

$$\begin{split} & \mathsf{D}_{(3,1)h}(Arch\_reln) \subseteq_h On, \\ & \mathsf{D}_{(3,2)h}(Arch\_reln) \subseteq_h On, \\ & \mathsf{D}_{(1,2)h}(Arch\_reln) \subseteq_h On, \\ & \mathsf{D}_{(1,2)h}(Arch\_reln) \subseteq_h Pouching \\ & \mathsf{D}_{(1)h}(Arch\_reln) \subseteq_h Pillar, \\ & \mathsf{D}_{(2)h}(Arch\_reln) \subseteq_h Pillar, \\ & \mathsf{D}_{(3)h}(Arch\_reln) \subseteq Block. \end{split}$$

 $D_{(i)h}$  and  $D_{(ij)h}$  are extensions of the image operator and select the *i*-th or the *i*-th and *j*-th arguments of a relation. The first relation expresses the fact that the third and first arguments of Arch\_reln are bounded by the On relation. Thus it is equivalent to

 $(y_1)(y_2)(y_3)[Arch\_reln(y_1, y_2, y_3) \supset On(y_3, y_1)].$ 

A pillar could be defined by:

 $Pillar =_h Stack \cup_h Block$ 

which would tie our definition back to that of Stack and the transitive closure of On.

A third capability that the extended system provides is a means of forming conjectures about sets of irreducible predicates, and thus meta-conjectures about the domain. For any set of known individuals or predicate names  $A = \{A_1, \ldots, A_n\}$ , we can form reducible sets  $A_r \subseteq A$  or irreducible sets  $A_i = (A_{i+}, A_{i-})$  where  $A_{i+}, A_{i-} \subseteq A$  and  $A_{i+} \cap A_{i-} = \emptyset$  (or, of course, sets of such sets, etc.). Thus, one may decide to specify

 $colour\_type = \{Red, \ldots, Violet\}$ 

where *Colour\_type* is introduced as an "internal" name, or by definition, based on the hypothesis:

Colour =  $_h Red \cup_h \cdots \cup_h Violet$ .

Similarly one may specify:

$$Bear\_type = (\{Black\_bear, Polar\_bear, \cdots\}, \{Red, Stack, \cdots\})$$

based on the hypothesis:

 $Black\_bear \cup_h Polar\_bear \cup_h \cdots \subseteq_h Bear.$ 

Bear\_type may be taken as irreducible, perhaps as a result of knowledge of individuals, for example the fact that some individual is known to be a bear, but is known to belong to

none of the known subgroups. Likewise, *colour\_type* may perhaps be taken as reducible as a result of pragmatic considerations. This time, for example, there is no known individual known to be coloured that isn't one of the known colours.

So the set axioms, as well as providing us with a more primitive basis for forming conjectures, also expand the expressiveness of the system. In particular we can relate some predicates on sets of objects to other predicates or individuals. Also we can form conjectures about other conjectures, or meta-conjectures. However, while the set of allowable individuals has been vastly expanded, the form of the conjectures is unchanged. The formal results of the last section still apply and, since in the intended model the property of being an individual is decidable, the overall system remains decidable.

#### 2.5. Comparison with Learning Systems

It is worth pausing at this point to compare the approach with other related work on learning in AI. Three systems are particularly relevant and serve to place the present work within the field. The early work of J.S. Brown [Brown 73] on automatic theory formation is a direct precursor to mine. Patrick Winston's dissertation [Winston 75] on learning structural descriptions from examples is a well-known early AI learning system and serves as a good representative of a general approach to learning from examples. Ehud Shapiro's work [Shapiro 81] on the inductive inference of theories from facts is similar to mine in broad outline, except that he makes substantial assumptions concerning how the domain of application is described.

The task of Brown's system is to propose definitions for a set of binary relations based on knowledge of the extensions of the relations. The system begins with a set of binary relations  $R = \{R_1, \ldots, R_n\}$  and a database containing all the tuples for which  $R_i(x, y)$  is true for each  $R_i \in R$ . The database is assumed complete and error-free. A process is given for proposing definitions of the relations. However the body of a definition is restricted to be either the composition of relations or disjunctions of such compositions. This format though is adequate for characterising a range of domains, including that of kinship relations.

The system is restrictive in that it demands complete and static data, and deals only with composition and disjunction. No analysis is carried out with regard to what may be conjectured, nor is an algorithmic analysis of the system given. Nonetheless the system illustrates some important points. Foremost is the overall difficulty in constructing *any* general system that will induce definitions from primitive instances. Given Brown's approach, and a domain as elementary as binary kinship relations, a vast number of conjectures was produced. However, as he remarks, it is not at all obvious which definitions should be selected as axioms and which should be left as redundant.

Brown's system is heuristic and was intended for direct implementation. Thus it dealt with matters such as efficiently searching for possible definitions, proposing definitions in a "simplest first" manner, etc. In contrast I have not addressed implementation issues, but rather have attempted to address general problems of hypothesis formation, and thus issues dealing with characterising a set of conjectures and maintaining the consistency of a set of conjectures.

Winston's program learns definitions of concepts in the blocks world domain by means of "successive refinements" based on carefully chosen examples. The program is given a training sequence of examples of the concept and of "near misses", where a near miss is an example that is quite similar to an instance of the concept but differs in a small number of significant details. Relevant features that the concept must have are extracted from the (positive) examples, while negative information is extracted from the near misses. The notion of near miss is centrally important, and serves to focus and accelerate the learning process. Winston is largely interested in the pragmatic aspects of a learning system, and concentrates on techniques that will speed the learning process or will assist in an implementation. In some sense then his approach is complementary to the one taken here (which avoids the more pragmatic issues). In addition, the set of sentences of HL subsumes the conjectures that may be formed in Winston's system<sup>5</sup>; thus anything that can be formed in his system can also be conjectured in HL. However, in HL the set of formable conjectures and the means of restoring consistency are precisely laid out whereas Winston does not address these issues.

Again, however, these aren't Winston's concerns; rather he deals with problems of implementation. Thus the problems of dealing with a vast number of possibilities for conjectures are avoided by means of a carefully selected sequence of examples and near misses. By restricting the language of discourse, the number of possibilities is further reduced. The result is that only a single (tentative) definition is retained. If worst comes to worst, and the definition becomes inconsistent, the system is able to resort to a backtracking strategy.

Shapiro's work is, superficially, the most similar to my own. Shapiro assumes that a domain is described by a stream of ground instances; based on the ground instances, a set of conjectured axioms for the domain is proposed and refined. The key difference is that in his approach far more is assumed about the way the domain is to be described. First, the domain is assumed to be describable by a set of rules in the form of restricted Horn clauses. Also the names of the classes for which defining axioms are to be induced are known beforehand and, for purposes of efficiency, each class is dealt with individually before the next class is considered.

A general, incremental algorithm for proposing a set of rules which imply the known ground instances is developed and described. The algorithm functions by successively reading ground instances. If the set of hypotheses is too strong (i.e. a contradiction is encountered), a contradiction backtracking algorithm is applied to locate the refuted hypothesis. The refuted hypothesis is then removed. If the conjectures are too weak, and don't imply the new ground instance, then a new conjecture is added, or a refinement of a previously refuted hypothesis is added. The algorithm has tuneable parameters that determine the complexity of the structure of a hypothesis and the complexity of derivations from the hypotheses.

The assumption that the domain is governed by rules allows an elegant, powerful, and reasonably efficient algorithm, and firm theoretical results concerning the types of languages that may be discovered. The system has, in addition to discovering an axiomatisation for simple arithmetic, synthesised logic programs for simple list-processing tasks, satisfiability of Boolean formulae, and tree isomorphism. These results though hinge on the assumption that the domain is describable in terms of (restricted) Horn clauses.

### 3. The Relation between Acquisition and Deductive Reasoning

#### 3.1. Reasoning with Knowledge and Conjecture

Given the preceding framework, arguably derived from first principles and with minimal *a priori* representational commitments, it is a reasonable question to ask how this approach could be used with some given representation scheme. In particular, consider now where we have a KB that consists of an arbitrary, consistent set of sentences, where some are known to be true, while others are hypothesised to be true. Since we require that the KB be able to reason about knowledge and hypothesis, this part of the problem either requires or presupposes a theory of incomplete knowledge. I have taken the latter course, and adopted the theory given in [Levesque 81]. This work presents a first-order logical language KL that

<sup>&</sup>lt;sup>5</sup> Winston actually gives a semantic net representation for his concepts. This representation however is clearly equivalent to a set of binary relations, and is useful mainly as a notational or implementational device.

can refer both to application domains and to what a knowledge base might know about such domains. KL is extended here to a language called HKL that is able to deal also with arbitrary conjectural sentences.

KL extends first-order predicate calculus (FOPC) by adding a sentential operator K, where K $\alpha$  can be read as " $\alpha$  is known to be true". HKL extends KL by the addition of a sentential operator H. where H $\alpha$  can be read as " $\alpha$  is conjectured to be true". Using HKL we can express sentences such as "John or Bill is hypothesised to be a teacher" or "it is known that Mary is hypothesised to be a teacher". HKL is specified as follows:

Axiom Schemata

- 1. The axioms of FOPC
- 2. K $\alpha$ , where  $\alpha$  is an axiom of FOPC
- 3.  $K(\alpha \supset \beta) \supset (K\alpha \supset K\beta)$
- 4.  $\alpha \equiv K\alpha$  where terms of  $\alpha$  are within the scope of a K or H operator
- 5. (x)K $\alpha \supset$ K $(x)\alpha$
- 6.  $K\alpha \supset (\neg H\alpha \land \neg H\neg \alpha)$
- 7.  $K(\alpha \supset \beta) \supset (\neg K\beta \supset (H\alpha \supset H\beta))$
- 8.  $H(\alpha \supset \beta) \supset (\neg K\beta \supset (K\alpha \supset H\beta))$
- 9.  $H(\alpha \supset \beta) \supset (\neg K\beta \supset (H\alpha \supset H\beta))$
- 10.  $((x)(K\alpha \lor H\alpha) \land (\exists x)\neg K\alpha) \supset H(x)\alpha$

#### Rules of Inference – Modus ponens and universal generalisation

Since the axiom  $K\alpha \supset \alpha$  is not a theorem of the system, what we are dealing with is not guaranteed to be true in the outside world, and thus is not knowledge *per se*, but rather corresponds to a notion of belief. Given this interpretation H $\alpha$  may be more appropriately read as " $\alpha$  is hypothesised to be true, under the assumption that the non-hypothesised sentences are in fact true". The distinction then is between potentially revisable statements, and statements which are taken to be knowledge, and thus true and unrevisable.

The first five axiom schemata and the rules of inference are those of KL. We obtain that knowledge and conjecture are closed under modus ponens, and meta-knowledge is complete and accurate. Also if something is known, neither it nor its negation is conjecture, and generalisation applies analogously to conjecture and knowledge. What this extension from KL to HKL buys us then is a means of distinguishing and reasoning with sentences known to be true, from those that are not known but are, for one reason or another, believed to be true.

In HKL the semantic interpretation of a closed formula depends on a world description (or interpretation in the Tarskian sense) and two descriptions of a KB, or model descriptions. The first model description gives the possible worlds that may be the case, given the known part of the KB; the second gives the subset of these possible worlds that may be the case assuming that the conjectured sentences are also true. From [Levesque 81] we have:

## Definition:

 $PRIM = \{p(n_1, \ldots, n_k) | n_i \in N \text{ and } p \in Pr(k)\}.$ 

WD =  $[PRIM \rightarrow \{T,F\}]$  the set of complete world descriptions.

 $MD = \{m | m \subseteq WD \text{ and } m \text{ is non-empty}\}\$  the set of model descriptions.

We can define an interpretation function for sentences of HKL as follows: Definition:

 $V \in [HKL \times WD \times MD \times MD \rightarrow \{T,F\}]$  is given by:

1)  $V(\alpha, s, m_1, m_2) = s(\alpha)$  if  $\alpha \in PRIM$ .

2)  $V(\neg \alpha, s, m_1, m_2) = T$  iff  $V(\alpha, s, m_1, m_2) = F$ .

- 3)  $V((\alpha \supset \beta), s, m_1, m_2) = T$  iff  $V(\alpha, s, m_1, m_2) = F$  or  $V(\beta, s, m_1, m_2) = T$ .
- 4)  $V((x) \alpha, s, m_1, m_2) = T$  iff for every n,  $V(\alpha[x/n], s, m_1, m_2) = T$ .
- 5)  $V(K\alpha, s, m_1, m_2) = T$  iff  $V(\alpha, s', m_1, m_2) = T$  for every  $s' \in m_1$ .
- 6)  $V(H\alpha, s, m_1, m_2) = T$  iff  $V(\alpha, s', m_1, m_2) = T$  for every  $s' \in m_2$  and  $V(\alpha, s', m_1, m_2) = F$  for some  $s' \in m_1$ .

**Definition:**  $\alpha \in$  HKL is valid iff for every s and  $m_1$  and  $m_2$ ,  $V(\alpha, s, m_1, m_2)$  is T and satisfiable if its negation is not valid.

The intuitive idea here is that whereas the first model description,  $m_1$ , specifies a set of complete world descriptions that may be the case, the second,  $m_2$ , restricts this set to a set of world descriptions that could be the case if the conjectured sentences actually turned out to be true. We obtain:

**Theorem:** A sentence  $\alpha$  of HKL is valid iff  $\alpha$  is a theorem.

#### 3.2. The Relation between HL and HKL

The logic HKL seems to have reasonable properties with respect to reasoning deductively with knowledge and hypothesis. There is a problem however with updating the KB. If the KB were told, for example, that P(a) was hypothesised but later that  $\neg P(a)$  was in fact the case, we would get an inconsistent knowledge base. Intuitively though, since a hypothesis is something that could conceivably be false, it seems that we should be able to "withdraw" a hypothesis. The difficulty of course is that HKL is monotonic and there is no obvious means of withdrawing a hypothesis.

A partial solution to the problem is to have the hypothetical portion of the KB, denoted  $KB_h$ , depend on the known portion  $KB_k$ , so that  $KB_h = HYP(KB_k)$  for some function HYP. For the rest of this section I review a version of the problem where the KB consists solely of known sentences and we want to "apply" HL to this KB to produce a hypothesised component. Thus for example if  $KB_k$  is

$$P(a), (x)(P(x)\supset Q(x)), R(a),$$

then applying HL to what is known about the ground instances could yield:

$$P = {}_{h}Q, Q = {}_{h}R, P = {}_{h}R$$

or the equivalent hypothetical KB in HKL

 $(x)(P(x)\equiv Q(x)), (x)(Q(x)\equiv R(x)), (x)(P(x)\equiv R(x)).$ 

The basic idea is that  $KB_k$ , which is expressed in HKL, determines a set of ground instances and a set of sentences that are representable in HL, and that by applying the procedure for restoring consistency to these sets we obtain a set of hypotheses following from this knowledge. The function HYP then takes a known KB and produces a hypothesised component  $KB_h$ , where  $KB_h$  is the result of "applying" HL to the original KB. In outline, we have the following procedure for forming a hypothetical component  $KB_h$  for a known KB:

Let  $KB = [KB_k, KB_h]$  be composed of the known and hypothesised portions of the KB, and assume initially that  $KB_h = \emptyset$ . The procedure is specified as follows:

- 1. Let  $\mathbf{G} = \{g \mid g \text{ is a ground atomic formula and } KB_k \vdash_{HKL} g\}$ .
- 2. Apply the procedure of section 2.3 to G to obtain a consistent set of conjectures C expressed in HL. The determination of new ground instances may also increase  $KB_k$ .
- 3. Let  $KB_h$  be the translation of the sentences of C into sentences of HKL; exclude any of those provable in  $KB_k$ .

11-15

Two issues concern us. The first is how HYP may be specified using HL. The second deals with possible restrictions that must be placed on  $KB_k$ . The first question is resolved

satisfactorily: there is generally a straightforward translation of sentences from HL to HKL and, given this, the function HYP can be specified in a straightforward manner. The only difficulty arises with conjectures involving sets of sets, where in some cases it may not be obvious how to express a conjecture in first-order terms.

The second question cannot be answered so satisfactorily. It proves to be the case that unless  $KB_k$  is equivalent to a set of ground instances, that applying HYP to it may result in inconsistency. The difficulty is that the procedure for restoring consistency relies on the existence of knowable ground instances whose truth value is unknown. However in FOPC it is possible to attribute a property to an unknown individual, and this attribution may lead to inconsistency here.

While this last result appears somewhat limiting, things in practice may not be too bad. Several considerations are relevant. First, the assumptions underlying HL are those that presumably underlie any coherent knowledge representation system: that there are individuals and relations. Given this, arguably the problems are intrinsic to the split between forming hypotheses and reasoning with knowledge, and thus, to the extent that the assumptions of HL are realistic, are unavoidable. Hence the problems addressed here arguably are the problems that must be addressed by any system that learns from examples, or else must be discharged by means of *a priori* decisions by the system designers. Second, while applying HYP to a general KB may lead to inconsistency, it need not necessarily do so. If it does, it may be possible that pragmatic considerations can be used to resolve or skirt a particular inconsistency.

#### 4. Conclusions

This work develops a formal, unified, and general (but basic) framework for investigating autonomous knowledge acquisition and acquisition systems. and their relation with knowledge representation systems. A primary goal was to keep the approach as general as possible and independent of any particular domain. representation scheme, or set of learning techniques. Hence, for example, there is no restriction placed on the ordering of the ground instances, nor is there any restriction with regard to introducing new ("known") predicate names during the learning process. The major methodological stance is the adoption of the same basic ontology — individuals and relations — common to representation systems, as a starting point for investigating acquisition. In this way factors arising from our orientation toward acquisition emerge, presumably unobstructed by any a priori representational commitments.

Formal systems are developed for introducing and maintaining the consistency of conjectures. An exact specification of what conjectures may potentially be formed is provided. Also it is shown how the consistency of a set of conjectures can be restored in the face of conflicting instances. The system illustrates that a reasonably rich and expressive set of conjectures can be derived using only a minimal set of assumptions. The expressiveness of the system is also indicated by the fact that it is as expressive as a number of existing AI acquisition systems, including [Brown 73], [Hayes-Roth 78], [Vere 78], and [Winston 75]. Results concerning decidability lend credence to the possibility that it may be efficiently implementable.

The framework presented is intended not only as a basis for the development of autonomous acquisition systems, but also as a neutral point from which such systems may be viewed and compared. Presumably the issues addressed here are common to, and are relevant to, any system for learning from examples. However only a set of formal issues have been addressed. Pragmatic issues concerned with the justification of conjectures, strength of evidence, and degrees of confirmation (to name a few) are outside the scope of this work. Also outside the scope of this paper is the question of exceptions to general statements; this area is addressed in [Delgrande 85] and [Delgrande 86].

The approach as it stands may have immediate practical applications. It could be used, for example, for verifying or completing definitional constraints in large databases or

knowledge bases. Thus, as a specific example, database systems often use integrity constraints to partly maintain consistency and reliability. However, given a large number of relations, it is an arduous task to specify all integrity constraints and to ensure that the set is consistent. The approach then seems suited to the task of automatically proposing and verifying such constraints.

The interaction between autonomous knowledge acquisition and deductive reasoning is also explored. The languages HL and HKL provide a possible logical framework for an integrated system for acquisition and reasoning. Given our assumptions, the natural direction for investigating acquisition from a stream of examples leads to a set-theoretic approach, while the natural direction for reasoning about relations among individuals leads to a first-order logical approach. However the interaction of these approachs leads to immediate (and, in hindsight, obvious) difficulties, which likely require informal, pragmatic techniques for their resolution.

#### Acknowledgment

This work is part of my doctoral dissertation at the University of Toronto, supervised by Prof. John Mylopoulos. Financial assistance from the Province of Ontario and the Department of Computer Science, University of Toronto, is gratefully acknowledged.

## Bibliography

- [1] A.R. Anderson and N.D. Belnap Jr., Entailment: The Logic of Relevance and Necessity, Vol. I, Princeton University Press, 1975
- [2] D. Angluin and C.H. Smith, "A Survey of Inductive Inference: Theory and Methods", Technical Report 250, Department of Computer Science, Yale University, 1982
- [3] J.S. Brown, "Steps Toward Automatic Theory Formation", Proceedings of the Third International Conference on Artificial Intelligence, Stanford, Ca., 1973, pp 121-29
- [4] H.B. Curry, Foundations of Mathematical Logic, McGraw Hill Co., 1963
- [5] J.P. Delgrande, "A Foundational Approach to Conjecture and Knowledge", Ph.D. thesis, Technical Report CSRI-173, Department of Computer Science, University of Toronto, September 1985
- [6] J.P. Delgrande, "A Propositional Logic for Natural Kinds", Sixth Conference of the Canadian Society for the Computational Studies of Intelligence, 1986
- [7] T.G. Dietterich, B. London, K. Clarkson and G. Dromey, "Learning and Inductive Inference", *The Handbook of Artificial Intelligence*, P.R. Cohen and E.A. Feigenbaum (eds.), William Kaufmann Inc., 1982
- [8] T.G. Dietterich and R.S. Michalski, "A Comparative Review of Selected Methods for Learning from Examples", in *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), Tioga, 1983
- [9] A.A. Fraenkel, Y. Bar-Hillel and A. Levy, Foundations of Set Theory, 2nd revised ed., North-Holland Pub. Co., 1973
- [10] E.M. Gold, "Language Identification in the Limit", Information and Control 10, 1967, pp 447-474
- [11] F. Hayes-Roth. "The Role of Partial and Best Matches in Knowledge Systems", in Pattern-Directed Inference Systems, D.A. Waterman and F. Hayes-Roth (eds.), Academic Press, 1978
- [12] G.E. Hughes and M.J. Cresswell, An Introduction to Modal Logic, Methuen and Col. Ltd., 1968.

- [13] J.A. Kalman, "Lattices with Involution", Transactions of the American Mathematical Society, vol. 87, 1958, pp 485-491
- [14] S.C. Kleene, Introduction to Metamathematics, North Holland Pub. Co., 1952
- [15] H.J. Levesque, "A Formal Treatment of Incomplete Knowledge Bases", Ph.D. thesis, Department of Computer Science, University of Toronto, 1981
- [16] R.S. Michalski, "A Theory and Methodology of Inductive Learning", in Machine Learning: An Artificial Intelligence Approach, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), Tioga, 1983
- [17] T.M. Mitchell, "Version Spaces: A Candidate Elimination Approach to Rule Learning", Proceedings of the Fifth International Conference on Artificial Intelligence, Cambridge, Mass., 1977, pp 305-310
- [18] J. Mylopoulos and H.J. Levesque, "An Overview of Knowledge Representation" in On Conceptual Modelling, M.L. Brodie, J. Mylopoulos, and J.W. Schmidt (eds.), Springer-Verlag, 1984
- [19] D.N. Osherson, M. Stob and S. Weinstein, "Formal: Theories of Language Acquisition: Practical and Theoretical Perspectives", Proceedings of the Eighth International Conference on Artificial Intelligence, Karlsruhe, West Germany, 1983
- [20] E.Y. Shapiro, "Inductive Inference of Theories from Facts", Research Report 192, Department of Computer Science, Yale University, 1981
- [21] R.G. Smith, T.M. Mitchell, R.A. Chestek and B.G. Buchanan, "A Model for Learning Systems", Proceedings of the Fifth International Conference on Artificial Intelligence, Cambridge, Mass., 1977, pp 338-343
- [22] E.M. Solway and E.M. Riseman. "Levels of Pattern Description in Learning", Proceedings of the Fifth International Conference on Artificial Intelligence, Cambridge, Mass., 1977, pp 801-11
- [23] S.A. Vere, "Inductive Learning of Relational Productions", in *Pattern-Directed Infer*ence Systems, Waterman and Hayes-Roth (eds.), Academic Press, 1978
- [24] P.H. Winston, "Learning Structural Descriptions from Examples" in The Psychology of Computer Vision, P. Winston (ed.), McGraw-Hill, 1975

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# KRITON:

# A Knowledge Acquisition Tool

# for Expert Systems

# Joachim Diederich

## Ingo Ruhmann

## Mark May

Research Division "Expert Systems" Institute for Applied Information Technology German Research Institute for Mathematics and Data Processing Schloß Birlinghoven Postfach 1240 D-5205 Sankt Augustin 1 West Germany Tel.: -2241/14-2687

# ABSTRACT

A hybrid system for automatic knowledge acquisition for expert systems is presented. The system integrates artificial intelligence and cognitive science methods to construct knowledge bases employing different knowledge representation formalisms. For the elicitation of human declarative knowledge the tool contains automated interview methods. The acquisition of human procedural knowledge is achieved by protocol analysis techniques. Textbookknowledge is captured by incremental content analysis. The goal structure of the knowledge elicitation methods is an intermediate knowledge representation language on which frame, rule and constraint generators operate to build up the final knowledge bases. The intermediate knowledge representation level regulates and restricts the employment of the knowledge elicitation methods. Incomplete knowledge is laid open by pattern directed invocation methods (the intermediate knowledge base watcher) triggering the elicitation methods to supplement the necessary knowledge.

Keywords:

expert systems, knowledge acquisition, cognitive science methods, intermediate knowledge representation.

## 1. Introduction

The KRITON system for knowledge acquisition is designed to meet the requirements of practical knowledge engineering tasks. Starting-point in developing the system was the assumption, that no single acquisition method will be powerful enough to overcome the so called knowledge acquisition bottleneck in knowledge engineering. To fill that gap, it requires *hybrid knowledge acquisition tools*, employing several knowledge acquisition methods to capture different kinds of human knowledge. Within the domain of expert systems, two major knowledge sources are available, in principle:

- The human expert with his declarative and procedural knowledge of the domain in question. This knowledge has been obtained in long practice and is often turned to account without sufficient meta-knowledge about the way it is used. To model problem solving processes mounting on these, often incomplete and unstructured, knowledge chunks is the task assigned to expert systems.
- 2. Well-structured static knowledge, fixed in the traditional mode of knowledge representation: natural language documents, text books, technical descriptions and instructions.

Depending on the actual application, all of the above mentioned knowledge sources may become important and a knowledge acquisition tool should be able to meet these requirements.

The aim of the present paper is to put forward an integrated methodological approach, which takes into consideration different types of expert knowledge (declarative knowledge vs. procedural knowledge) combining so far divergent methods to a modular knowledge acquisition system, each submethod being able to acquire information on specific aspects of the problem solving process and to transform the gained information into a knowledge representation formalism.

The KRITON-approach for automatic and semi-automated knowledge acquisition integrates methods of artificial intelligence with those of cognitive science. One of the important strategies of knowledge engineering is the interview, i.e. the dialogue between knowledge engineer and expert to inquire about important terms and concepts of an application domain (NEWELL & SIMON 1972) and their interdependence.

From cognitive science we adopted the method of protocol analysis, i.e. processing and transformation of texts gained by transcribing protocols of loud thinking during a problem solving process. In AI, the analysis of thinking-aloud protocols has been automated quite early (WATERMAN & NEWELL 1971, 1973).

The analysis of texts with respect to syntactic, semantic and pragmatic criteria also goes back to cognitive science. Although content analysis has developed into a standard method in the social sciences, it still represents a not much used option for knowledge acquisition on the basis of natural language texts. KRITON uses a form of incremental content analysis to take advantage of these valuable knowledge sources.

Figure 1 shows the basic architecture of the KRITON system. In short: three knowledge elicitation methods are employed, namely an automated interview, content analysis and

protocol analysis. After a completion process and a consistency check the elicited information is transformed into an intermediate knowledge representation language consisting of a descriptive language for functional and physical objects, and a propositional calculus.



Frame, rule and constraint generators operating on the intermediate representation level are finally used to build up the destination knowledge base. On the other side, the already acquired knowledge guides the employment of the elicitation methods to complete the knowledge bases incrementally.

## 2. Methods for Knowledge Elicitation

On its first processing level our model makes use of three different knowledge elicitation methods, to be outlined in this chapter.

## 2.1. Interview

One of the most important strategies of knowledge engineering is the interview. GROVER (1983) distinguishes four different interview techniques for rule acquisition:

1. Forward scenario simulation

An applicational situation within a problem domain is selected and investigated under laboratory conditions. The expert reports on the relevant terms and concepts and describes the steps in problem solving, i.e. his or her own reasoning to achieve a goal.

- 2. Goal decomposition The knowledge engineer divides the overall problem into subgoals and asks the expert to describe paths for achieving the subgoals.
- 3. Procedural simulation

GROVER (1983) uses this umbrella term for protocol analysis. In his opinion controlling interventions by the knowledge engineer are absolutely necessary.

4. Pure reclassification

Expert statements are further differentiated and classified into specific objects and relations between objects by means of a dialogue between knowledge engineer and expert. As a result of the interview, object-relations may be reclassified and new taxonomic relations eventually discovered. An interview techniques not mentioned in GROVER's classification is

5. Laddering

The expert is asked to name important concepts of the problem domain in question. These concepts are then used as basis for the interview to follow. Especially supertypes and instances of generic concepts are inquired about, allowing the derivation of a taxonomic structure. An automated version of this interview techniques is implemented in ETS (BOOSE 1984, 1985).

# 2.1.1. Interview Methods in KRITON.

In the KRITON system, interview techniques are completely automated, what is to say, the expert interacts directly with the system. A combination of the repertory grid technique, forward scenario simulation and laddering is used to explore the relevant concepts of a problem domain.

The top level technique is the repertory grid approach: triples of semantic related concepts are presented to the expert in form of natural language sentences and the expert is asked for attributes two of the concepts share distinguishing them from the third.

If the expert is not able to name discriminating attributes, the system switches into laddering mode to explore taxonomic relations between the concepts. The expert may either answer with a single word, denoting a specific concept, or can enter free text, which is analyzed through morphologic-syntactic techniques to detect the relevant concepts.

The interview produces structured objects at the intermediate knowledge representation level. These objects incorporate the explored taxonomic relations and attributes.

# 2.2. Protocol Analysis

Protocol analysis refers to the automated or semi-automated analysis of thinking-aloud protocols, that is, tape-recorded utterances of an expert during a problem solving episode. The result of the protocol analysis can be considered as a path through successive knowledge states representing the sequence of the problem solving events. In case that an expert system uses this sequence of knowledge states (e.g. in consultation) a surface modeling of the human problem solving process takes place.

Although automatic protocol analysis has been suggested as an adequate method for knowledge acquisition in expert systems for some time now, fully developed systems are rare. A consistent approach to protocol analysis is described by KUIPERS & KASSIRER (1983, 1984), their approach aiming at both, a structural description of the problem domain and a qualitative simulation of the transitions between knowledge states during the problem solving process. A constraint language is used to fill up incomplete protocol segments.

The power of protocol analysis quite decisively depends on the quality of the protocol recording. Only if the protocol is actually one of loud thinking during a problem solving process and only if this protocol has been correctly transcribed, automatic analysis will be successful. As the success of any protocol analysis depends on the quality of the recorded information, detailed and adequate instructions with respect to the attainment of protocol recordings of verbal utterances during the problem solving process are of great importance. In any case, it requires psychologically trained manpower to achieve a constant cognitive load of the thinking-aloud expert (for a comprehensive review on problems with verbal data, see ERICSSON & SIMON 1980, 1984).

Granularity of expert knowledge has turned out to be a serious and not easy to handle problem. Even the most careful employment of protocol analysis will not avoid problemirrelevant knowledge elements to be acquired. As soon as not directly problem-relevant concepts are uttered by the expert, they are contained in the verbal material and hence are fed into the analysis. For example, this can be the case when the expert starts commenting upon, explaining or evaluating his thoughts or actions.

The other extreme, however, might as well occur, namely an expert communicating his "compiled knowledge" to the system. This is to say, that the expert over his extended learning process has combined inference steps so that the verbal report on the problem solving process is incomplete. The expert skips, more or less small, nonetheless essential inference steps. Even if this does not necessarily affect the efficacy of the future expert system, it will reduce the explainability of the problem solving process.

# 2.2.1. Protocol Analysis in the KRITON System

Protocol analysis as a knowledge elicitation method is used in the KRITON system in order to get hold of procedural human knowledge. Ideally, knowledge that was part of content analysis or some previous interview is observed "in action" during the protocol recording. Goal structure of the protocol analysis is the propositional part of the intermediate knowledge representation level. In KRITON protocol analysis is accomplished in five steps. First the transcribed protocol is partitioned into segments on the basis of the experts speech pauses during recording. The second step is the semantic analysis of the segments, creating propositions for each segment. In a third step, the appropriateness of the selected operators and arguments is checked upon. Next, a knowledge base matching is attempted to instantiate variables inside the propositions (variables are inserted if appropriate references for pronouns etc. cannot be found). In a last step, propositions are arranged according to their apparence in the natural language protocol.

## 2.3. Content Analysis

Knowledge engineering phase models recommend the knowledge engineer to start off with studying manuals and documents on the problem domain in question. This can be very time-consuming, particularly if the knowledge engineer is supposed to become an expert on the topic before beginning his or her actual work.

For about forty years content analysis has been concerned with analyzing texts, especially newspaper articles. Since the 50ies, programs for automatic content analysis are available (KRIPPENDORFF 1980; MERTEN 1983). Utilization of these methods for constructing knowledge-based systems have, in the best case, been outlined in the published literature. NISHIDA, KOSAKA & DOSHITA (1983), for example, analyze hardware manuals by means of action-event models. FREY, REYLE & ROHRER (1983) use discourse representation structure (DRT, by H. Kamp) as "intermediate level" between the natural-language text (a fragment of German language) and a data basis.

## 2.3.1. Content Analysis in KRITON.

KRITON supports the knowledge engineer in incremental content analysis. The knowledge engineer can ask for statistical information on keyword frequencies in a selected text. If a text seems expedient for knowledge acquisition, the user can define the size of a textfragment surrounding the keywords, to be used for the generation of basic propositions in a similar manner as in protocol analysis.

The resulting propositional structures are sometimes faulty and therefore not appropriate for inference processes. The goal structures as part of the intermediate knowledge representation are to be constructed in an interactive process, where possible objects and relations are presented to the user in a menu and window system. Appropriate items can be selected by mouse-operations and the corresponding knowledge structures are set up.
# 3. Intermediate Knowledge Representation Level

In our system, all output from the mentioned above techniques is translated into an intermediate knowledge representation system. This representation system has two subparts: a descriptive language for functional and physical objects. representing the generic concepts, and a propositional calculus representing the transformation path of these concepts during the human problem solving process.

The general architecture of this representation level is similar to KL-TWO (VILIAN 1985), that means, a terminological component is used to represent conceptual knowledge and a propositional component is introduced to handle complex statements about the world.

The description language consists of structured objects, their features and interrelations in a semantic net. The semantic net is the goal language for the methods interview and content analysis and serves as the basis for the frame generation process.

The second part of the intermediate knowledge representation language is a propositional calculus, using semantic primitives to describe the basic relations of concepts detected by protocol analysis. The set of semantic primitives is not complete and will have to be updated for each application domain (e.g. technical applications).

The intermediate knowledge representation level allows integration of different knowledge sources and supplies the tool with openness towards elicitation methods currently not available. Moreover, it can be used for the generation of various knowledge bases for different expert system shells and knowledge representation systems taking advantage of the facilities of interactive knowledge base generation.

4. Dealing with Incomplete Knowledge: Knowledge Guided Knowledge Elicitation.

The use of knowledge elicitation methods depends not only on decisions of the knowledge engineer but also on requirements the KRITON system detects on the basis of the already acquired knowledge.

A significant role in dealing with *incomplete knowledge* is played by the *watcher*. The watcher is an always active demon controlling the intermediate knowledge representation for missing components. For example, the user (the knowledge engineer or the expert) might have generated several objects during the incremental content analysis without any relation to the taxonomic organization of the objects of the corresponding domain (i.e. no information about the inheritance paths, part-of relations or instance relations was given). The watcher checks all objects at the intermediate knowledge representation level for missing, but possible or indispensable, links (every object has to be placed in a taxonomic organization), sends a message to the user and recommends the employment of an elicitation method to complete the knowledge base. The watcher is also invocated if an elicitation method starts, informing the user about incomplete parts of the knowledge base. Furthermore, the user can delegate the selection of concepts to be used in an interview to the watcher. In this case the demon looks for semantic related but incomplete objects and triggers an interview further exploring that domain.



### 5. Knowledge Base Generation

As mentioned above, the intermediate knowledge representation is the blackboard for frame, rule and constraint generation.

The task of the frame generator is to translate the information stored in structured objects and their relation into a frame language. In principle, this is a simple syntactic transformation process. After frame generation, the user can interactively correct the translation process with a structure editor.

The output of the protocol analysis is the input for the rule generator. A subset of propositional clauses, extracted from adjacent segments in the thinking-aloud protocol, is offered to the user for rule generation. The user can either reject the proposal or use it for rule generation. Rule junctors and rule actors are inserted by selection from pop-up-menus, premises and actions by entering the corresponding proposition number. Again, a rule editor can be called to correct for shortcomings of the automated protocol analysis.

Thus far, frame and rule generator build knowledge bases using the BABYLON frame and rule language (DI PRIMIO & BREWKA 1985).

If the user, through interaction with the system, detects global value restrictions, the constraint generator is used to represent these global restrictions in a constraint language (see GUESGEN 1986).

# 6. Steps in Knowledge Engineering with KRITON.

The following is a description of phases in automatic knowledge acquisition using the KRI-TON methodology. These steps are not strongly chronological. Especially through the influence of the knowledge-guided knowledge elicitation process, loops (cyclic and alternating employment of the different KRITON submethods) are probable and for applications of considerable size can be considered necessary.

There is no doubt, that in certain cases the exclusive employment of single submethods of KRITON will be successful.

The technique of incremental content analysis will be described in more detail in forthcoming publications. The overall knowledge acquisition process consists of three levels: knowledge elicitation, intermediate knowledge representation and knowledge base generation.

1. Definition of the domain.

The actual knowledge domain defined by the situation, in which human problem solving process occurs, is initially investigated by means of interview techniques. The definition of the domain and the breaking down of the experts extensive knowledge into proportionate subparts is an important precondition for the automated acquisition process.

II. Elicitation of declarative knowledge by automated interview techniques and incremental content analysis.

The important terms and concepts of a concrete knowledge domain to be investigated by means of automated protocol analysis or other acquisition methods for procedural knowledge are inquired about and entered into the computer-based analysis system. Interview and content analysis are employed in a cyclic manner until the network of structured objects reaches a significant size.

III. Guided protocol recording

A protocol of loud thinking is tape-recorded. This requires a careful guidance to secure constant verbalization of the expert during the problem solving work. It will take quite a few protocols if the problem domain is not to be restricted to a single problem solving path.

### IV. Transcription

The recorded protocol is transcribed. While punctuation is not used, speech pauses are supplemented. The protocol is entered into the analysis system.

V. Protocol segmentation

The protocol is automatically divided into numbered segments, the speech pauses determining the length of the segments.

VI. Search for the knowledge elements in the segmented protocol

The segmented protocol is searched for the various knowledge elements of the problem space. Concepts that are detected are stored together with the segments they are contained in.

# VII. Semantic analysis of the segmented protocol

By comparison with the available lexicon entries, all words contained in the segments found by V. are examined whether they include

- a) ordinal relations, e.g. A is smaller than B or X is equal to Y or,
- b) tendencies, e.g. The state of X is stable. The value of Y continues to increase.

### VIII. Propositionalization

If such elements are found, the knowledge elements together with the operators interconnecting them are set down in a propositional calculus.

# IX. Completion of propositions

The system tries to find knowledge elements, which allow a completion of the above mentioned propositions. This is first done within the same segment, subsequently in the neighboring segments.

X. Knowledge base matching.

The method described under VIII. does not allow the identification and solution of references, especially over longer distances. In case of a proper realization of protocol recording, however, complex syntactic constructions are not to be expected. By way of trial, the completion of propositions is accomplished by searching for complete propositions displaying the components already extracted. The missing arguments are taken from these propositions.

# XI. Intermediate knowledge representation

All output from the protocol analysis is integrated in the intermediate knowledge representation system. This language supplies a propositional language as a goal for the protocol analysis. Each proposition consists of an operator, i.e. a semantic deep case representing the basic relation between the acquired concepts, a segment marker, i.e. a pointer to the origin of the proposition in the natural language protocol, and the relevant concepts.

# XII. Checking for completeness in the network of structured objects.

Usually, protocol analysis will exhibit voids in the network of structured objects. This is always the case, when concepts used during the thinking-aloud procedure are not sufficiently defined. In this case, interview and content analysis should be repeated.

XIII. Frame generation

Structured objects in the semantic net of the intermediate knowledge representation level are translated into frame format, i.e. the BABYLON frame representation language. In general, frame generators for several other languages can be added, using the intermediate knowledge representation as a blackboard.

### XIV. Rule generation

Rule generation is an interactive process realized by mouse-operations for the selection of propositions to be used on the left or right side of rules. Corrective actions can be taken by calling a structure editor. The organization of rule sets as well as the selection of control strategies, for the present, remains a task of the knowledge engineer.

# XV. Constraint generation.

If global dependencies between data are discovered while using KRITON, these relations are encoded in a constraint language. The selection of data and their relations proceeds by mouse-interaction.

# 7. Comparison with other Systems of Automatic Knowledge Acquisition

None of the so far developed systems for automatic knowledge acquisition has reached product features. Most systems are experimental in character. Though the approaches are often very different and based on different theoretical assumptions, some common features are identifiable. The efforts center around the construction of a conceptual structure by means of an interactive system.

Nevertheless, there is no system that handles multiple knowledge representation formalisms or makes use of several different knowledge sources to acquire both declarative and procedural knowledge. Some systems, for instance ROGET, have no elicitation component and are primarily used for purposes of knowledge base extension.

KADS (BREUKER & WIELINGA 1985) is an interactive system using a set of different functions in support of the knowledge engineer. This includes assistance in planning problems, data interpretation and consistency check. KADS was mainly based on a KL-ONE implementation in PROLOG provided with a simple rule interpreter, the rules being part of a network. This knowledge based system, containing task-dependent and domainindependent information, is used for the interactive analysis of a knowledge domain. Analysis is controlled by interpretation models that are typical for specific problem solving processes (e.g. in a diagnostic task domain). In principle, KADS can be considered as a library containing different elicitation methods.

ROGET (BENNETT 1985) directly interacts with an expert to construct a rule base that is understood as fundamental conceptual structure of the knowledge domain. ROGET itself was developed in the context of EMYCIN and supports only systems of this type. A ROGET consultation is used for the following tasks:

- 1. definition of problem solving type;
- 2. acquisition of conceptual structure;
- 3. analysis of conceptual structure;
- 4. operationalization of conceptual structure for a specific knowledge engineering environment (of the EMYCIN type).

ETS (BOOSE 1984, 1985) is another interactive system for generation of knowledge bases of the EMYCIN and OPS 5 types. The heart of the system consists of an on-line implementation of the repertory grid test by KELLY (1955). Factor analytical methods are

applied for an investigation of semantic distance between concepts and possible implicational relations. ETS only supports rule base generation.

#### 8. Implementation

So far, all components described in this paper are implemented in a preliminary form on a XEROX-1108 machine in INTERLISP-D using object-oriented features from LOOPS. The structured object representation of the intermediate knowledge representation language is realized in form of LOOPS-objects. Protocol and content analysis make use of a lexicon for "closed class" words. Detection of word-stem works by means of an analysis of inflection, which itself is part of the lemmatization component. Lemmatization is to some extent lexicon-based and partly rule-based. The hit rate is well above 90 percent and in so far comparable to that of other systems. A first application of the system is planned for the 2nd half of 1986. Application domain is an expert system for planning and configuration of bureau equipment.

### 9. Conclusions

We are aiming at an integrated, modular system-tool for knowledge acquisition in expert systems. On the one hand the system should be of high supportive value for the acquisition of declarative and procedural expert knowledge. On the other hand it should be open, in the sense, that it provides facilities for its own extension and elaboration. At the present stage of the systems development the protocol and the content analysis still work inaccurate and sometimes erroneous. For the present, these shortcomings in automatic text analysis are compensated by the use of appropriate editors, through which the employment and testing of the system in applied industrial fields is guaranteed.

To sum up: the here presented approach to automated knowledge acquisition not only has the potency of taking advantage of developments in hybrid knowledge representation formalisms, but also is hybrid in the sense that it makes use of information from different knowledge sources. In our opinion, the guaranteed openness for future extensions is a most promising feature of the knowledge acquisition tool presented here, supplying it with a remarkable amount of applicability for various industrial fields.

### 10. References

- Bennett, J.S. ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System. Journal of Automated Reasoning, 1, 49-74, 1985
- Boose, J. Personal Construct Theory and the Transfer of the Human Expertise. In: Proc. of the National Conference on Artificial Intelligence. Austin, Texas 1984
- Boose, J. A Knowledge Acquisition Program for Expert Systems based on Personal Construct Psychology. International Journal of Man-Machine Studies, 23, 495-525, 1985
- Breuker, J. & Wielinga, B. KADS: Structured Knowledge Acquisition for Expert Systems. Proc. Expert Systems and their Applications, Vol. 2, 887-900, 1985.

- Di Primio, F. & Brewka, G. BABYLON: Kernel System of an Integrated Environment for Expert System Development and Operation. Proc. of the Fifth International Workshop "Expert System and their Applications", Avignon, France, May 1985
- Ericsson, K.A. & Simon, H.A. Verbal Reports as Data. Psychological Review, 87, 3, 1980
- Ericsson, K.A. & Simon, H.A. Protocol Analysis. Verbal Reports as Data. The MIT Press, Cambridge, Mass. 1984
- Frey, W., Reyle, U. & Rohrer, C. Automatic Construction of a Knowledge Base by Analyzing Texts in Natural Language. IJCAI 83, 727-729, Karlsruhe 1983
- Grover, M.D. A Pragmatic Knowledge Acquisition Methodology. IJCAI 83, 436-438, Karlsruhe 1983
- Guesgen, H.W. Consat-0. Foundations of a System for Constraint Satisfaction. Unpublished manuscript, St. Augustin, 1986
- Kelly, G. The Psychology of Personal Constructs. New York: Norton, 1955
- Krippendorff, K. Content Analysis. An Introduction to its Methodology. Beverly Hills, 1980
- Kuipers, B. & Kassirer, B. How to Discover a Knowledge Representation for Causal Reasoning by Studying an Expert Physician. IJCAI 83, 49-56, Karlsruhe 1983
- Kuipers, B. & Kassirer, B. Causal Reasoning in Medicine: Analysis of a Protocol. Cognitive Science, 8, 363-385, 1984
- Merten, K. Inhaltsanalyse. Westdeutscher Verlag, Opladen 1983
- Newell, A. Simon, H.A. Human Problem Solving. Prentice-Hall Inc., Englewood Cliffs, N.J. 1972
- Nishida, T., Kosaka, A. & Doshita, S. Towards Knowledge Acquisition from Natural Language Documents — Automatic Model Construction from Hardware Manuals — IJCAI 83, 482-486, Karlsruhe, West Germany, 1983
- Vilian, M. The Restricted Language Architecture of a Hybrid Representation System. IJCAI 85, 547-551, Los Angeles 1985
- Waterman, D.A. & Newell, A. Protocol Analysis as a Task for Artificial Intelligence. Artificial Intelligence, 2, 285-318, 1971
- Waterman, D.A. & Newell, A. PAS II: An Interactive Task Free Version of an Automatic Protocol Analysis System. IJCAI 1973. Menlo Park, Ca.: Stanford Research Institute, 431-445, 1973

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# MOLE: A Tenacious Knowledge Acquisition Tool

# Larry Eshelman, Damien Ehret, John McDermott, and Ming Tan Department of Computer Science Carnegie Mellon University Pittsburgh, Pennsylvania 15213

# Abstract

MOLE can help domain experts build a heuristic problem-solver by working with them to generate an initial knowledge base and then detect and remedy deficiencies in it. The problem-solving method presupposed by MOLE makes several heuristic assumptions about the world, which MOLE is able to exploit when acquiring knowledge. In particular, by distinguishing between covering and differentiating knowledge and by allowing covering knowledge to drive the knowledge acquisition process, MOLE is able to disambiguate an under-specified knowledge base and to interactively refine an incomplete knowledge base.

# 1. Introduction

MOLE [Eshelman 86] is an expert system shell that can be used in building systems that do heuristic classification. It is both a performance system which interprets a domain dependent knowledge base and a knowledge acquisition tool for building and refining this knowledge base. MOLE the performance system presupposes that the task can be represented as a classification problem: some object is selected from a set of pre-enumerable candidates (e.g., faults, diseases, components) on the basis of weighted evidential considerations (e.g., symptoms, cues, requirements) [Clancey 84, Clancey 85, Buchanan 84]. MOLE the knowledge acquisition tool builds a knowledge base by eliciting knowledge from the domain expert guided by its understanding of how to represent the knowledge required by its problem-solving method, and discovers missing knowledge and refines the knowledge base guided by its understanding of how to diagnose what knowledge the problem-solving method might be missing.

MOLE belongs to a family of knowledge acquisition tools which get their power by paying close attention to the problem-solving method used by their performance systems [McDermott 86, Gruber 86]. Examples of such systems are TEIRESIAS [Davis 82], ETS [Boose 84], MORE [Kahn 85a, Kahn 85b], KNACK [Klinker 86], SALT [Marcus 85], and SEAR [vandeBrug 85]. MOLE resembles the first three systems in that its method can be described as a variant of heuristic classification. It differs from them in that its problem-solving method incorporates certain explicit assumptions about the world which, along with several assumptions about how experts express themselves, are exploited during the knowledge acquisition process.

Our goal has been to make MOLE smart -- i.e., to enable it to build a reasonable knowledge base with a minimal amount of information elicited from the expert. Our research strategy has been to overly restrict (or so it seems at first) the information that MOLE can elicit from the expert and then to search for heuristics that will enable it to build from this limited information a knowledge base that can

perform the given task reasonably well. In this paper we shall describe the results of this strategy. In Section 2 we describe the method presupposed by MOLE and the set of knowledge roles imposed by the problem-solving method. In Section 3 we describe how MOLE's knowledge of its performance system's method is exploited in knowledge acquisition.

# 2. MOLE's Method

MOLE the knowledge acquisition tool gets its power from its knowledge of the problem-solving method of MOLE the performance system. In this section we shall describe the method used by MOLE the performance system. MOLE's problem-solving method is a variant of heuristic classification. MOLE selects or classifies candidate hypotheses on the basis of "rules" associating the candidates with various evidential considerations. However, unlike some heuristic classification systems such as MYCIN, these rules are not just arbitrary implications among arbitrary facts about the world [Szolovits 78]. By making certain assumptions about the world and interpreting the rules or associations in light of these assumptions, MOLE is able to obtain considerable leverage for its knowledge acquisition. MOLE's problem-solving method consists of the following steps:

- 1. Ask what symptoms need to be explained
- 2. Determine what hypotheses will explain or cover these symptoms (COVERING-KNOWLEDGE)
- 3. Determine what information will differentiate among the hypotheses covering any symptom (DIFFERENTIATING-KNOWLEDGE)
- 4. Ask for that information
- 5. If any differentiating knowledge needs to be explained, go to 2
- Pick the best combination of viable hypotheses that will explain all of the symptoms (COMBINING-KNOWLEDGE)
- 7. If there is information that will affect the viability of some combination of hypotheses, ask for that information, and go to 2
- 8. Display the results

Central to MOLE's method is the distinction between evidence that needs to be explained or covered by some hypothesis and evidence that helps differentiate among hypotheses. The former, covering knowledge, reflects two basic assumptions that MOLE makes about the world:

- 1. Exhaustivity: every abnormal finding has an explanation -- i.e, some candidate hypothesis will account for it.
- 2. Exclusivity: explanations should not be multiplied beyond necessity -- i.e., do not accept two hypotheses if one will do.

The exhaustivity assumption enables MOLE to interpret the hypothesis-symptom associations in its domain model causally. Every symptom is assumed to have a cause. If a symptom is not explained by one hypothesis, it must be explained by another. The exclusivity assumption is a version of Occam's razor: All other things being equal, parsimonious explanations should be favored. As a general rule of thumb, the types of events represented by hypotheses are fairly rare, so it is unlikely that several occur simultaneously. (Of course, two such events might be interrelated, but then this should be represented by combining knowledge.)

An important corollary follows from the exhaustivity and exclusivity assumptions: Accept the best candidate relative to its competitors -- i.e, a candidate may "win" by ruling out competing candidates. Because symptoms must be explained by some hypothesis (exhaustivity), one of the hypotheses must be true. And because only one hypothesis is likely to be true (exclusivity), we can drive up the support of one hypothesis by driving down the support of its competitors or vice versa. This provides

the basis for MOLE's second kind of knowledge role, differentiating knowledge.

**Differentiating knowledge** enables MOLE to evaluate the relative likelihood of hypotheses explaining the same symptom. MOLE understands four types of differentiating knowledge:

- Anticipatory knowledge
- Circumstantial knowledge
- Refining knowledge
- Qualifying knowledge

In all four cases, the knowledge helps differentiate which hypothesis is the most likely explanation of a symptom. Anticipatory knowledge is closely tied to covering knowledge. Given the covering knowledge that event E<sub>1</sub> explains event E<sub>2</sub>, anticipatory knowledge is the additional information that the presence of E<sub>1</sub> is likely to lead to E<sub>2</sub>, or alternatively, that the absence of E<sub>2</sub> tends to rule out E<sub>1</sub>. Circumstantial knowledge, like covering knowledge, associates evidence with hypotheses. But unlike covering knowledge, the evidence does not have to be explained or covered. Circumstantial knowledge is merely correlated -- positively or negatively -- with the hypothesis. Such evidence indicates that the hypothesis is more or less likely to be true, but there is no presumption that one of the hypotheses with which it is associated must be true. Refining knowledge points to distinguishing features of a symptom which indicate that a proper subset of the hypotheses covering it are more likely to contain the correct explanation. By refining the symptom, MOLE is able to differentiate among the hypotheses that might explain the unrefined symptom. Qualifying knowledge is any background condition which qualifies the strength of an association. A qualifying condition may either strengthen an association or weaken (or completely mask) an association.

Covering knowledge and differentiating knowledge are local to a symptom. Combining knowledge, on the other hand, is global. Once the hypotheses which best cover or explain each symptom that needs to be covered have been discovered, combining knowledge is used to select the best combination of hypotheses that will explain all the symptoms that are present. More specifically, combining knowledge overrides MOLE's default method for combining hypotheses. MOLE classifies active hypotheses in one of three categories: accept, reject, or indeterminate. Hypotheses that are clearly better than any of their competitors at explaining some symptom are tenatively accepted. Hypotheses which are not accepted and are not needed to explain any symptoms because these symptoms are explained by some accepted hypothesis are rejected. All other active hypotheses are classified as indeterminate. Among the indeterminate hypotheses MOLE tries to find the smallest viable set that will explain all the unexplained symptoms. Combining knowledge overrides MOLE's default procedure and recommends a less parsimonious combination of hypotheses.

The ability to make use of combining knowledge means that MOLE is not simply limited to selecting hypotheses but can also construct hypotheses. However, since its constructive ability is rudimentary, we have continued to present MOLE as only appropriate to tasks that are amenable to heuristic classification. MOLE's method still places strong limitations on the type of tasks for which it would be appropriate.

We have mainly used MOLE to build knowledge bases for diagnostic tasks. However, we have described MOLE's method so that no particular type of domain is presupposed, for we suspect that there is a much wider range of tasks for which MOLE would be useful. For example, MOLE might be used in a domain whose task is component selection. Covering knowledge would link the components (hypotheses) with requirements that must be met (symptoms). Differentiating knowledge would be information indicating what tradeoffs can be made between various components. Combining knowledge would be heuristics suggesting how to select a combination of components if no single component met a set of overlapping requirements.

Since most of our research has involved diagnostic problems, we will illustrate MOLE's method and the knowledge roles presupposed by its method with an example of a system which diagnoses problems in a coal burning power plant. The knowledge base built by MOLE the knowledge acquisition tool enables MOLE the performance system to diagnose problems connected with the boiler, the central unit in a power plant. The problems rarely prevent the boiler from functioning, but they are a major source of inefficiency. A boiler that is functioning inefficiently can waste millions of dollars of fuel as well as dump tons of pollutants into the atmosphere.

Suppose, for example, the operator notices that there is a loss in gas and that the ash leaving the boiler is dark. These are two symptoms that need to be covered or explained. The loss in gas has two potential explanations: (1) a high gas temperature; (2) high excess air. The other symptom, dark ash, has three potential explanations: (1) high excess air; (2) low excess air; (3) large fuel particles. For each symptom present, MOLE tries to differentiate among the potential causes. For example, when there is high excess air, there is a strong expectation (anticipatory knowledge) that there will be a high fly ash flow. If this expectation is not met, support for the high excess air hypothesis is driven down. Consequently, support for the competing explanation, high gas temperature, is driven up and becomes the likely explanation for the loss in gas. Since high excess air is also a potential explanation for the dark ash, support for its competitors, low excess air and large fuel particles, are also driven up. MOLE tries to further differentiate between these two hypotheses. For instance, if the flame temperature is not low, contrary to what one would expect if there is low excess air, then support for the low excess air hypothesis is also driven down, leaving large fuel particles as the most likely explanation for the dark ash. Once MOLE has finished locally differentiating the hypotheses covering each symptom that is present, MOLE looks at what combination of hypotheses will best explain all the symptoms. In this case MOLE will accept two hypotheses, high gas temperature and large fuel particles, since the only hypothesis which will explain all the relevant symptoms has been ruled out by the absense of a high fly ash flow. Having accepted these two hypotheses, MOLE next sees whether they, in turn, can be treated as symptoms that are explained by higher level hypotheses. In this case, they both have higher level explanations. For example, the large fuel particles can be explained by either the setting of the pulverizer or by a malfunction of the pulverizer. Knowing that the grinder had recently been maintained would provide provide circumstantial evidence that a malfunction is unlikely and thus drive up the likelihood that the setting of the pulverizer is the explanation for the large fuel particles. MOLE continues until it has attempted to explain every event that can be explained.

In the next section we will draw from our experiences with the power plant diagnosis task to illustrate how MOLE uses its knowledge of its method to guide the knowledge acquisition process both when generating the initial knowledge base and when refining it.

# 3. Knowledge Acquisition

Usually it is not very hard to elicit knowledge from an expert. The hard problem is eliciting the right sort of knowledge. Knowledge needs to be in such a form that it will be applied to the problem in the right way at the right time. The first step in satisfying this need is to explicitly identify the appropriate problem-solving method for the task and the types of knowledge roles relevant for this method. Once this is done, it is fairly easy to build a knowledge collector. However, if the ultimate goal is to replace the knowledge engineer with an automated system, rather than providing the knowledge engineer with a programming tool, then two troublesome features of the knowledge acquisition process must be addressed:

- Indeterminateness: When specifying associations between events, the expert is likely to be fairly vague about the nature of these associations and events.
- Incompleteness: The expert will probably forget to specify certain pieces of knowledge.

The indeterminateness problem reflects the fact that experts are not accustomed to talking about the associations between events in a way that precisely fits the problem-solving method's predefined knowledge roles. Since our ultimate goal is to develop a knowledge acquisition tool that replaces the knowledge engineer, the burden is upon the knowledge acquisition system to make sense of whatever information the domain expert is willing to provide. Although the expert can be encouraged to be as specific as possible, a smart knowledge acquisition tool must be able to tolerate ambiguity and indeterminateness.

The incompleteness problem is the problem of how to identify missing or incorrect knowledge. The expert, no matter how qualified and thorough he may be, is going to forget to mention certain special circumstances. And sometimes the expert will make mistakes. Thus, a smart knowledge acquisition tool needs to be able to incrementally add knowledge to the knowledge base, refine existing knowledge, and sometimes correct existing knowledge.

The indeterminateness problem and the incompleteness problem dominate the two phases of knowledge acquisition: (1) the gathering of information for constructing the initial knowledge base and (2) the iterative refinement of this knowledge base. During the first phase, MOLE mainly relies upon static techniques of analysis. MOLE examines specific associations and events in light of the context provided by the surrounding structures. During this phase MOLE concentrates on disambiguating the information provided by the expert, although MOLE also tries to recognize areas where the knowledge is obviously incomplete. During the second, dynamic phase, MOLE and the expert interact in order to refine the knowledge base. The expert gives MOLE a test case and tells MOLE the correct diagnosis. If MOLE the performance program comes to an incorrect conclusion, MOLE the knowledge acquisition tool tries to determine the source of the error and recommends possible remedies. Typically, this means adding knowledge or qualifying existing knowledge, but sometimes the interpretation provided in the previous phase needs to be revised.

In the following two subsections, we shall discuss the techniques MOLE uses during both the construction and refinement phases of knowledge acquisition.

#### 3.1. Constructing the Initial Knowledge Base

MOLE initiates the knowledge acquisition process by asking the expert to list the events -- i.e., hypotheses and evidence -- that are commonly relevant to the expert's domain and to draw associations between pairs of events. This information is easy to elicit from the expert, although the expert will often overlook certain associations or events. However, in order for MOLE to be able to fashion this network of events and associations into a knowledge base suitable for diagnosis, four additional pieces of information are needed:

- 1. The type of event (i.e., whether observed or inferred)
- 2. The type of evidence an association provides (e.g., covering evidence, anticipatory evidence)
- 3. The direction of an association (e.g., does  $E_1$  explain  $E_2$  or vice versa)
- 4. The numeric support value attached to an association

MOLE understands that experts are not very good at providing such information and so does not require that the expert provide a fully specified network of associations. Instead, MOLE relies on its expectations about the world and how experts enter data in order to mold an under-specified network of associations into a consistent and unambiguous knowledge base.

The most critical piece of information is the association type. Both the direction and support value of an association are dependent upon its type. The problem with directly asking the expert for the

association type is that it is hard to convey to the expert what these types mean. The distinction between covering and circumstantial evidence, for example, is part of MOLE's jargon, not the expert's. But the problem is not simply a matter of finding a translation from MOLE's jargon into terminology that the expert can understand. The indeterminacy is deeper than this. Not only does the expert have difficulty distinguishing among types of evidence, often he can't decide whether an event should be classified as evidence rather than a hypothesis. He may not even be sure whether he should say it is observed or inferred. Perhaps it can be observed, but only with difficulty. For example, one of the causes of a misbalance of convection is the presence of fouling. Whether the operator can see signs of fouling depends upon what part of the boiler it is occuring. On those occasions when he cannot observe fouling, he still may be able to infer fouling. Furthermore, even if an event is observed, it can function as a hypothesis. Suppose that fouling can be observed and that a low heat transfer needs to be explained. A misbalance of convection competes with a misbalance of radiation as explanations of a low heat transfer. If fouling is discovered before the explanation of a low heat transfer has been resolved, then fouling serves as evidence for a misbalance of radiation. On the other hand, if it has already been ascertained that there is a misbalance of radiation then the fouling provides the explanation.

After several unsuccessful attempts to find ways of unambiguously eliciting the association types from the expert, we discovered that we were looking for the solution in the wrong place. In the three domains that we have explored most extensively the only associations that the experts have ever spontaneously entered are covering and anticipatory associations. Instead of having to worry about identifying half a dozen knowledge roles, we only have to worry about two. In a sense there is no identification problem at all, since every anticipatory association can be interpreted as a covering association and vice versa. However, the anticipatory strengths of some covering associations are so weak that they can be ignored in practice. So the problem is to identify which covering associations provide significant anticipatory evidence. We shall return to this question when we discuss how MOLE determines the support values of associations.

On reflection, it isn't very surprising that the initial knowledge acquisition process generates just these two types. The starting point in diagnosis is a set of events that need to be explained. MOLE asks for the names of these events and their potential explanations, and then for explanations of these explanations. The knowledge acquisition process is driven by covering knowledge and to a lesser extent anticipatory knowledge. Other types of knowledge are secondary and tend to be overlooked by the expert. For example, given that low excess air needs to be explained, the most common explanations will occur to the expert -- e.g., the fan power has been exceeded or there is a leak in the duct. While he is thinking about fan power it may occur to him that if he knows that a valve is malfunctioning, he will anticipate that the fan power has been exceeded. On the other hand, it is unlikely to occur to him that the installation date of the duct is a relevant circumstance for determining the likelihood of a leak. When providing covering knowledge the expert is biased toward thinking of events causing other events. He tends to overlook circumstances or states that are correlated with an event except in the context of some actual test case.

Given the expert's bias toward providing only covering associations, finding out the direction of these covering associations would seem to be a straightforward matter. All we need to do is ask the expert whether  $E_1$  explains  $E_2$  or vice versa. Unfortunately, about twenty percent of the time experts reverse the direction. When the expert says that  $E_1$  explains  $E_2$  what he may mean is that  $E_1$  explains why he would accept  $E_2$ , not that  $E_1$  is the physical explanation of  $E_2$ . The same confusion occurs with cause and effect language. Sometimes the expert will say that  $E_1$  causes  $E_2$  and mean that the occurrence of  $E_1$  "caused" him to think of  $E_2$  as the explanation of  $E_1$ . We have found that the most reliable directional information that can easily be elicited from the expert is an association's temporal direction -- i.e., which event occurs first. MOLE assumes, all things being equal, that the temporal direction is a reliable indicator of the explanatory direction. If, for example, the expert says that  $E_2$ 

supports  $E_1$  and that  $E_1$  precedes  $E_2$ , MOLE assumes that  $E_1$  explains  $E_2$ . However, if the expert indicates that he is not sure about the temporal direction, MOLE will try to elicit more information in order to clarify the nature of the association. It could be that the association is circumstantial, or, more likely, that one event is really a direct indication of the other event. For instance, a small, red flame is a direct indication of a low flame temperature. Although MOLE interprets the flame temperature as explaining the flame size and color, the expert may be reluctant to put a temporal direction on this relationship.

The final type of information that MOLE needs in order to fashion the initial network of associations into a knowledge base is each association's support value. However, experts do not like providing numeric support values and are not very good at it. In our experience, when experts are asked to indicate the degree of support of some piece of evidence for a hypothesis by selecting a number within a fixed range, they will think about the question for awhile and then almost inevitably choose some number near the middle of the range. Fortunately, it turns out that the support values do not have to be very accurate, and that on the basis of a few simple assumptions MOLE can assign default support values that are just as good, if not better, than those assigned by the expert. (They subsequently can be adjusted by MOLE during dynamic analysis.)

MOLE's method for assigning support values for covering evidence follows directly from MOLE's exhaustivity and exclusivity assumptions. The assumption that every symptom can be explained by some hypothesis plus the assumption that only one of the covering hypotheses is likely to be the explanation suggests that the positive support provided by the presence of a symptom should be distributed among the hypotheses and should sum to 1.0. Since MOLE initially has no information how this support should be distributed, it makes the default assumption that the support values for any symptom should be equally divided among the hypotheses which explain it.

The method for assigning support values for anticipatory evidence has a weaker rationale. MOLE assumes that if an expert spontaneously mentions a piece of anticipatory knowledge, then it is the sort of information that is likely to have a significant impact; thus, a relatively high support value is assigned. The problem, then, is to determine when an expert is spontaneously entering anticipatory evidence. As was explained earlier, covering evidence and anticipatory evidence are closely linked. They enable MOLE to reason in both directions between two events. If  $E_1$  explains  $E_2$ , then the occurrence of  $E_2$  provides some evidence for  $E_1$ . If in addition there is a strong anticipatory connection between the events, then the occurrence of  $E_1$  increases the likelihood of  $E_2$ . MOLE pays attention to the direction of this supporting relationship. If the expert indicates that  $E_1$  supports  $E_2$  and that  $E_1$  is temporally prior, then MOLE will infer that the expert is spontaneously entering an anticipatory association. MOLE will also assume that there is a covering association between the two events. On the other hand, if the temporal and supporting directions are different -- e.g., if  $E_1$  supports  $E_2$ , and  $E_2$  is prior to  $E_1$  -- MOLE assumes that the expert is entering a covering association, and assumes, until evidence to the contrary, that there is no strong anticipatory association.

MOLE has one other opportunity to identify anticipatory associations in the initial phase of knowledge acquisition. MOLE monitors the static knowledge base and notes any symptoms which have hypotheses that cannot be differentiated. MOLE assumes that if the expert provides several explanations for some event, then he probably can provide knowledge that will differentiate among them. For example, suppose that MOLE learns during the initial construction of the knowledge base that low excess air can be explained by either the fan power being exceeded or by a leak in the duct, but is not told about any evidence that would help differentiate between these hypotheses. MOLE will ask the expert how he can tell which hypothesis is the cause of low excess air. The expert may indicate that if there is no loud noise coming from the wind box, then MOLE can rule out that there is a leak. This will enable MOLE, at least in principle, to differentiate the two hypotheses.

Before discussing the second phase of knowledge acquisition, something should be said about the absence of any stress on qualifying knowledge during the first phase. Based on the descriptions of other diagnostic systems, one might expect that for a system to perform adequately most associations would need a large number of qualifications. If MOLE did not distinguish between covering and differential knowledge and simply tried to build rules associating evidence with hypotheses, then, no doubt, its initial network of associations would require many qualifications. However, because of the emphasis that MOLE puts on covering knowledge, MOLE tends to build multi-layered networks with sparse connections rather than flat networks where several layers of information need to be compiled into one rule. By asking for the explanations of a symptom rather than the conditions for accepting a hypothesis, MOLE naturally discovers intermediate events which mediate between the bottom level symptoms and the top level hypotheses. The knowledge base for diagnosing boiler problems, for example, consists of eight levels of explanation and has as much depth as it has breadth.

However, there is a price to pay for the heavy reliance on intermediate events. There is the danger that redundant pathways are drawn between events. MOLE needs to check for duplicate pathways. For example, in one session MOLE was told that the two explanations for a misbalance of convection are fouling and low excess air. However, in a later session MOLE learned that low excess air can be explained by the fan power being exceeded which, in turn, can be explained by fouling. Thus, there is a direct path from fouling to misbalance of convection and an indirect path through fan power being exceeded and low excess air. The question is whether the direct path is an actual causal path or whether it represents a compiled version of the longer path. MOLE has no way of knowing the answer, but it is important that MOLE notice such occurrences and ask the expert about them.

#### 3.2. Refining the Knowledge Base

In the previous section, we mentioned that MOLE checks to make sure that hypotheses can in principle be differentiated. If they cannot, MOLE tries to elicit knowledge from the expert that will enable it to differentiate among them. However, such static techniques of refinement are quite limited. The expert typically needs a richer context to remind him of missing knowledge, and MOLE needs a richer context in order to be able to distinguish a wider range of association types. The needed context is provided by feedback from the expert during dynamic analysis. The expert gives MOLE a test case and tells MOLE the correct diagnosis. If MOLE the performance system has come to an incorrect diagnosis, MOLE the knowledge acquisition tool tries to determine the source of the errors and recommends possible remedies.

In the remainder of this section we will discuss MOLE's techniques for doing dynamic analysis. These techniques reflect the three types of knowledge roles understood by MOLE: covering knowledge, differentiating knowledge, and combining knowledge.

#### 3.2.1. Differentiating Knowledge

If MOLE's diagnosis does not match that supplied by the expert, MOLE first determines whether or not the diagnosis would have been reachable if the hypotheses had been differentiated differently. If it is so reachable, then MOLE looks for missing differentiating knowledge. MOLE next decides whether there is a global or a local problem. In a local problem support needs to be shifted among hypotheses which explain the same symptom. In a global problem the raising or lowering of some hypothesis's likelihood will affect the explanations of several symptoms in the right direction and so have a global effect. (Sometimes what looks like a global problem is actually a collection of several local problems.)

Anticipatory knowledge: If the problem is global, MOLE will first focus on identifying anticipatory knowledge. As was explained in our discussion of static analysis, if the expert spontaneously enters a covering association, MOLE cannot tell whether the two events are also linked by an anticipatory

association, so it only generates a covering association. MOLE now examines whether there are any inactive covering associations which would have the desired effect if they were also interpreted as anticipatory assumptions.

For example, knowing that a loss in gas is explained by either a high gas temperature or high excess air, and that a high fly ash flow is explained by either very small fuel particles or high excess air, and not having any information to differentiate between these pairs of hypotheses, MOLE will accept high excess air as the explanation for both symptoms and reject the other two hypotheses. MOLE's reasoning is that since it does not have any reason to favor the other two hypotheses over high excess air, high excess air is the best choice because it alone can explain both symptoms. However, when given this case, the expert told MOLE that it was wrong -- it should have rejected high excess air and accepted high gas temperature and very small fuel particles. MOLE recognized that if there was some evidence that could drive down the likelihood of high excess air, then the likelihood of the other two hypotheses would be driven up. Noting the absence of an event which provides covering evidence (a high oxygen reading) which would have been explained by high excess air, MOLE reasoned that if these two events were also linked by anticipatory knowledge, then the absence of a high oxygen reading would tend to rule out the high excess air hypothesis. MOLE asked the expert whether the absence of a high oxygen reading ruled out high excess air. He confirmed that it did.

**Circumstantial knowledge**: If the problem is global and no anticipatory knowledge is discovered, MOLE focuses on circumstantial knowledge. One reason that circumstantial knowledge is given less priority than anticipatory knowledge is that we have found that anticipatory knowledge is more common. But a more important reason is that anticipatory knowledge is tied to covering knowledge and so the existing covering knowledge provides the richer context which MOLE tries to exploit first. Even when MOLE interprets a piece of knowledge as circumstantial, it does so tentatively, keeping in mind that in another example it may learn that it should be interpreted as anticipatory knowledge.

In our previous example, if MOLE had not known anything about the relevance of a high oxygen reading and the expert had entered it for the first time, then MOLE would have interpreted it as circumstantial evidence. If MOLE later learned that a high oxygen reading needs to be explained by high excess air, then it would add a covering association and reinterpret the circumstantial association as an anticipatory association.

Refining knowledge: If the problem is local to a symptom, then MOLE will ask the expert whether there are any features of the symptom which narrow the set of the hypotheses that are the likely cause.

So far in the power plant domain we have not found any occasion for refining a symptom. However, in other domains refinement has been useful. For example, in a system that diagnoses automobile problems the symptom that the car won't start was refined to include the feature of whether or not the engine cranked. This feature is used to distinguish the hypothesis that the battery is dead from alternative explanations for the car not starting such as being out of gas, faulty spark plugs, and carburetor problems.

Qualifying knowledge: Qualifying knowledge can adjust existing anticipatory, circumstantial, and refining knowledge. If such knowledge exists but is in the wrong direction, or in the right direction but is too weak, MOLE asks if there is any background condition which would strengthen or weaken the existing association in the right direction. In addition qualifying knowledge can be added in conjunction with an anticipatory association. This, in effect, adds a condition which strengthens a potential anticipatory association into one that has a significant impact.

For example, when trying to acquire knowledge that would lead it to accept that the fan power is

being exceeded as the explanation for low excess air, and so reject a leak in an air duct as the explanation, MOLE looked for potential anticipatory knowledge. It knew that there was evidence of fouling and that this could cause the fan power to be exceeded, but discovered that the link was too weak to have the needed effect. This suggested to MOLE that it should ask the expert whether there was any condition which would make it very likely that the fan power would be exceeded when there was fouling. The expert confirmed that fouling almost always leads to the fan power being exceeded when there is a heavy load.

Adjusting support values: If the expert fails to provide any special knowledge which will differentiate among the hypotheses in the needed fashion, then there are three possibilities: (1) There is a need for combining knowledge. (2) The numeric support values are wrong and need to be adjusted. (3) A combination of locally differentiating evidence is needed, instead of a single globally differentiating piece of knowledge. MOLE's choice of what to explore first will depend on the "maturity" of the network and the reliability of feedback from the expert. It is easier for the expert to understand and respond to questions about locally differentiating evidence, so this option is usually tried first, provided the network is mature.

However, early in the refinement phase the most likely possibility is that the support values are wrong. After all, the support values are simply MOLE's guesses based on the structure of the network. The most common change is a shift of a symptom's support from some of its hypotheses to others. For example, there are only two hypotheses explaining the fan power being exceeded -- fouling and a malfunction of a valve -- so MOLE initially set the support value of each association to 0.5. However, fouling is much more likely than a valve malfunction, unless the fan has not been maintained for a long time. In a test case where the fan power had been exceeded and there was no further evidence for favoring fouling over a malfunction of a valve, MOLE could not decide between the two hypotheses. The expert indicated, however, that MOLE should have picked fouling. After failing to elicit any knowledge from the expert that would help it differentiate between these two causes, MOLE decided that its apportionment of support between the two hypotheses must be mistaken and shifted support from the malfunction of a valve hypothesis to the fouling hypothesis.

#### 3.2.2. Covering Knowledge

So far we have examined the case were MOLE would have reached the correct diagnosis if it had differentiated among the hypotheses differently. If differentiation is not the problem, then MOLE looks for missing covering knowledge. There are two possible cases: (1) a hypothesis should be rejected but can't be because it is needed to explain some otherwise unexplained symptom; (2) a hypotheses should be accepted, but it was rejected because it is not needed to explain any symptom.

If a hypothesis fails to be rejected because it is needed to explain some symptom, then MOLE asks the expert for an alternative explanation for the symptom. For example, MOLE discovered that low excess air was a possible explanation of dark ash after being told in a test case that both of the hypotheses that it considered possible explanations for the dark ash should be rejected. MOLE reasoned that dark ash must have an explanation that it did not know about, and asked the expert for this explanation.

But suppose the expert had told MOLE that there is no alternative explanation; yet, the two explanations that it knows about are incorrect. MOLE considers two possibilities: (1) the symptom report is mistaken; (2) the evidence is not really a symptom -- i.e., it has misinterpreted the event as covering evidence. MOLE would first inquire whether it is possible that the reported observation that there is dark ash could be be mistaken. If it can be, MOLE would lower the default certainty for the report of this event. Then if it is again faced with a situation where there is evidence against both the explanations for this symptom, it will reject them both and suggest that a mistaken report (or mistaken observation) is the most likely explanation for the reported symptom. On the other hand, if the expert

is quite certain about this reported event, then MOLE would examine how it can most coherently reinterpret the association -- e.g., interpret the association as a circumstantial association instead of a covering association.

If a hypothesis is mistakenly rejected because it is not needed to explain anything, then MOLE asks the expert if there is some symptom that is present which can be explained by this hypothesis. For example, MOLE learned about the high stack gas temperature reading in a test case where it wrongly rejected high gas temperature as an explanation for loss in gas. MOLE the performance system had reasoned that since high excess air was needed to explain high fly ash and could also explain loss in gas, the high gas temperature was not needed. However, upon being informed that it should not have rejected high gas temperature, MOLE the knowledge acquisition tool reasoned that there must be some piece of evidence that high gas temperature explained which was not explained by any other hypothesis. MOLE asked the expert for this information, and was told that the missing piece of evidence was the fact that the stack gas temperature reading was high.

#### 3.2.3. Combining Knowledge

But what if the expert indicates that there is no missing evidence that needs to be explained by the rejected hypothesis? MOLE reasons that it is probably missing some combining knowledge. If its hypotheses are properly differentiated locally and there is no missing piece of evidence for the hypothesis to explain, then this must be a case where the best explanation for some symptom is the combination of several hypotheses. It should be stressed that MOLE's default strategy will accept several explanations for a set of symptoms, provided each hypothesis is the best explanation of at least one symptom. What distinguishes the case where combining knowledge is needed is that several hypotheses are needed to explain one symptom. MOLE, with the guidance of the expert, acquires a rule for handling this special case.

For example, a misbalance of radiation and a misbalance of convection are alternative explanations for low heat transfer. Furthermore, the misbalance of convection hypothesis is only needed to explain the low heat transfer. When MOLE has reason to accept a misbalance of radiation as the explanation of low heat transfer, its default combining strategy will dictate that it reject the misbalance of convection hypothesis as unneeded. When told that this diagnosis was wrong, MOLE first looked for some symptom which a misbalance of convection explains but a misbalance of radiation doesn't. Upon learning that there was no such symptom, MOLE reasoned that it must be missing a special combining rule, and asked the expert for the circumstances that lead him to accept both misbalance of radiation and misbalance of convection. In this case, it is the presence of low excess air.

As MOLE has evolved, dynamic analysis has increasingly taken on a more important role. By not insisting that the expert identify an event's type or an association's direction during the construction phase if he is uncertain about its value, the knowledge base used in the refinement phase is less determinate than it would otherwise be. These indeterminate associations provide MOLE with valuable pieces of information when doing dynamic analysis. Whenever MOLE learns that it has made an incorrect diagnosis and has located the portions of the network where there is likely to be missing knowledge, indeterminate associations in these parts of the network are prime candidates for the missing knowledge. And since MOLE knows something about these associations -- e.g., what events they connect -- it can be quite certain how they should be interpreted even if the expert isn't.

# 4. Conclusion

MOLE illustrates how much power a knowledge acquisition tool can obtain from a set of domain independent heuristics about the knowledge acquisition process and the nature of the world as it relates to diagnosis. MOLE plays the role of an experienced knowledge engineer who is able to work

in conjunction with a domain expert and build a diagnostic system, even though the knowledge engineer has little or no knowledge of the domain. Like such a hypothetical knowledge engineer MOLE begins with no knowledge of the target domain nor any understanding of the domain's vocabulary. By interpreting its assumptions about the world in terms of explicit knowledge roles that guide heuristic classification and by exploiting a few heuristics about how domain experts are likely to express themselves, MOLE is able to intelligently extract from the expert information relevant for building a reasonable knowledge base for performing the given diagnostic task.

# Acknowledgements

We want to thank Gary Kahn and Sandra Marcus for helpful suggestions in the development of MOLE. We also would like to thank Holger Sommer for serving as MOLE's expert in the power plant domain and for the useful feedback that he provided in the development of MOLE.

# References

[Boose 84]	Boose, J. Personal construct theory and the transfer of human expertise. In Proceedings of the National Conference on Artificial Intelligence. Austin, Texas, 1984.
[Buchanan 84]	Buchanan, B. and E. Shortliffe. Rule-based Systems: the Mycin experiments of the Stanford Heuristic Programming Project. Addison-Wesley, 1984.
[Clancey 84]	Clancey, W. Classification problem solving. In Proceedings of the National Conference on Artificial Intelligence. Austin, Texas, 1984.
[Clancey 85]	Clancey, W. Heuristic classification. Artificial Intelligence 27, 1985.
[Davis 82]	Davis, R. and D. Lenat. Knowledge-Based Systems in Artificial Intelligence. McGraw-Hill, 1982.
[Eshelman 86]	Eshelman, L. and J. McDermott. MOLE: a knowledge acquisition tool that uses its head. In Proceedings of the National Conference on Artificial Intelligence. Philadelphia, Pennsylvania, 1986.
[Gruber 86]	Gruber, T. and P. Cohen. Design for Acquisition: Designing Knowledge Systems to Facilitate Knowledge Acquisition. Technical Report, University of Massachusetts at Amherst, 1986.
[Kahn 85a]	Kahn, G., S. Nowlan, and J. McDermott. Strategies for knowledge acquisition. IEEE transactions on Pattern Analysis and Machine Intelligence 7(5), 1985.

[Kahn 85b]	Kahn, G., S. Nowlan, and J. McDermott. MORE: an intelligent knowledge acquisition tool. In <i>Proceedings of Ninth International Conference on Artificial Intelligence</i> . Los Angelos, California, 1985.
[Klinker 86]	Klinker, G., J. Bentolila, S. Genetet, M. Grimes, and J. McDermott. <i>KNACK - report-driven knowledge acquisition</i> . Technical Report, Carnegie-Mellon University, Department of Computer Science, 1986.
[Marcus 85]	Marcus, S., J. McDermott and T. Wang. Knowledge acquisition for constructive systems. In Proceedings of Ninth International Conference on Artificial Intelligence. Los Angelos, California, 1985.
[McDermott 86]	McDermott, J. Making expert systems explicit. In Proceedings of 10th Congress of the International Federation of Information Processing Societies. Dublin, Ireland, 1986.
[Szolovits 78]	Szolovits, P. and R. Patil. Categorical and probablilistic reasoning in medical diagnosis. <i>Artificial Intelligence</i> 11, 1978.
[vandeBrug 85]	van de Brug, A., J. Bachant, J. McDermott. Doing R1 with Style. In Proceedings of the Second IEEE Conference on Artificial Intelligence Applications. Miami, Florida, 1985.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# AN OVERVIEW OF KNOWLEDGE ACQUISITION AND TRANSFER

Brian R Gaines Department of Computer Science, University of Calgary Calgary, Alberta, Canada T2N 1N4

Abstract: A distributed anticipatory system formulation of knowledge acquisition and transfer processes is presented which provides scientific foundations for knowledge engineering. The formulation gives an operational model of the notion of expertise and the role it plays in our society. It suggests that the basic cognitive system that should be considered is a social organization, rather than an individual. Computational models of inductive inference already developed can be applied directly to the social model. One practical consequence of the model is a hierarchy of knowledge transfer methodologies which defines the areas of application of the knowledge engineering techniques already in use. This analysis clarifies some of the problems of expertise transfer noted in the literature, in particular, what forms of knowledge are accessible through what methodologies. The model is being used as a framework within which to extend and develop a family of knowledge support systems to expedite the development of expert system applications.

### INTRODUCTION

Growing recognition of the significance of knowledge-based computing systems has focused attention on processes of knowledge acquisition and transfer. Commercial application of expert systems is being impeded by the knowledge engineering bottleneck and has led to the development of rapid prototyping tools. These have proved practically useful but the range of applications of existing tools is limited, and it is not clear what limitations are inherent in the prototyping tools, what arise from the shells and their knowledge representation and inferencing procedures, and what arise from our lack of understanding of the underlying processes of knowledge acquisition and transfer.

Knowledge acquisition and transfer processes in human society are complex and poorly understood. Many disciplines are involved in their study but none provides a comprehensive framework let alone overall theory. Neurology, psychology, linguistics, education, sociology, anthropology, philosophy and systems theory all have significant contributions to make. However, integrating them, disambiguating jargon, and combining different objectives and perspectives are major intellectual problems. The task is attracting increasing effort but this is unlikely to have significant short-term inpact on knowledgebased system development.

Knowledge acquisition and transfer are now important aspects of advanced computer technology, and neither the pragmatic development of rapid prototyping techniques nor the dependence on partial theories from other disciplines can be regarded as satisfactory foundations. We need to be eclectic in acquiring techniques from all possible sources, but we also need to complement our pragmatism with effort to develop scientific foundations for our activities. We cannot be satisfied until we have a precise understanding of the processes underlying expertise, its operation, acquisition and transfer. This is not a simple requirement since it entails understanding the nature of knowledge, its dynamics and application. The foundations of computer technology in the physical sciences are no longer adequate and need extension into the humanities. The philosophy, psychology and sociology of knowledge processes are highly significant to future computing, and we have to operationalize theories and obtain precise answers to questions that have long been regarded as certainly controversial and possibly intractable.

This paper sketches a framework for encompassing the variety of processes and phenomena of knowledge acquisition and transfer that provides a convenient classification, an indication of overall theoretical foundations, and a guide to research methodologies.

# PHENOMENA OF EXPERTISE ACQUISITION AND TRANSFER

The problem of expertise elicitation from a skilled person is well-known in the literature of psychology (Nisbett & Wilson 1977, Broadbent, Fitzgerald and Broadbent 1986). Hawkins (1983) has analysed the nature of expertise and emphasizes its severe limitations and dependence on critical assumptions which are often implicit. Dixon (1981) has surveyed studies showing that much human activity is not accessible to awareness. Collins (1985) has studied knowledge transfer processes among scientists and suggests that some knowledge may not be accessible through the expert, not only because he cannot express it, but also because he may not be aware of its significance to his activity. Bainbridge (1979, 1986) has reviewed the difficulties of verbal debriefing and notes that there is no necessary correlation between verbal reports and mental behavior, and that many psychologists feel strongly that verbal data are useless. Clinical psychologists see the problem as one of cognitive defences that impede internal communication, and have developed techniques of verbal interaction to identify underlying cognitive processes (Freud 1914, Kelly 1955, Rogers 1967). These can be used to by-pass cognitive defences, including those resulting from automization of skilled behavior. Hayes-Roth, Waterman & Lenat (1983) have given guidelines about knowledge transfer from experts to expert systems, and Welbank (1983) has surveyed the psychological problems of doing this.

The main problems identified in accessing an expert's knowledge are:-

- "Expertise" may be fortuitous. Results obtained may be dependent on features of the situation which the expert is not controlling.
- Expertise may not be available to awareness. An expert may not be able to transmit the expertise by critiquing the performance of others because he is not able to evaluate it.
- Expertise may not be expressible in language. An expert may not be able to transmit the expertise explicitly because he unable to express it.
- Expertise may not be understandable when expressed in language. An apprentice may not be able to understand the language in which the expertise is expressed.
- Expertise may not be applicable even when expressed in language. An apprentice may not be able to convert verbal comprehension of the basis of a skill into skilled performance.
- "Expertise" expressed may be irrelevant. Much of what is learnt, particularly under random reinforcement schedules, is superstitious behavior that neither contributes nor detracts from performance.
- Expertise expressed may incomplete. There will ususually be implicit situational dependencies that make explicit expertise inadequate for performance.
- "Expertise" expressed may incorrect. Experts may make explicit statements which do not correspond to their actual behavior and lead to incorrect performance.

In the development of knowledge engineering methodologies and rapid prototyping techniques for expert systems the emphasis has been on interviewing experts, and hence on linguistic trasmission of expertise. A rich variety of alternative methods for expertise transfer exists in human society:-

- Expertise may be transmitted by managing the learning environment. A trainer may be able to establish effective conditions for an apprentice to acquire expertise without necessarily understanding the skill or himself being expert in it.
- Expertise may be transmitted by evaluation. A trainer may be able to induce expertise by indicating correct and incorrect behavior without necessarily understanding the skill in detail or himself being expert in its performance.
- Expertise may be transmitted by example. An expert may be able to transmit a skill by showing his own performance without necessarily understanding the basis of his expertise.

There are many sources of expertise that do not involve others:-

- Expertise may be acquired by trial and error learning. This is the basic inductive knowledge acquisition process that is always in operation although heavily overlaid by the social transfer processes already discussed.
- Expertise may be acquired by analogical reasoning. The transfer of models and skills from one situation to another is an important source of knowledge.
- Expertise may be acquired by the application of general laws and principles to new situations. The use of physical laws and systemic principles to generate specific expertise is the basis of scientific and engineering expertise.

These many aspects of the problems of knowledge acquisition and transfer are further confounded by the combinatorial possibilities resulting from the recursive nature of knowledge processes. We can acquire knowledge about knowledge acquisition. We can mimic the behavior of an expert coach managing a learning environment. We can give evaluative feedback on analogical reasoning or the application of principles and laws. We can express the principles behind the effective management of learning in specific domains. The knowledge processes of human society are rich and complex, and, in practice, many of these possibilities will be instantiated as parallel, inter-related activities in any knowledge acquisition and transfer situation.

# SOCIO-SYSTEMIC FOUNDATIONS OF EXPERTISE

Figure 1 shows some of the relations between a knowledge engineer, or knowledge transfer system, an expert in a problem domain, and the physical, social, knowledge and problem environments.



Figure 1 Relations between knowledge engineer, expert and environments

An "expert role" has been shown rather than an expert person to emphasize that expertise is focused on specific task domains and is not an indication of global capability. Within that role various significant psychological processes have been noted: perception, action and language as interfaces with the world; memory, awareness, intentions, emotions and attention as associated internal phenomena. Some of these processes go beyond the basic information processing models of current cognitive science (Norman 1980), but they are essential to knowledge acquisition and transfer.

The specific problem environment for the expert is shown as intersecting three general environments that are common to most problems: physical, social and knowledge. The knowledge engineer is part of the social environment but outside the problem environment. He interacts with the expert and also with his environments. For example, if the expert is performing a task in the physical environment then the knowledge engineer will attempt to understand the basis of that task, observe the expert performing it, and may manage the task environment to create scenarios in which to observe specific features of the skill.

The three environments correspond to Popper's (1968) basic delineation of 3 worlds:-

- World 1, the physical environment, is the world of 'things', of physical objects and their dynamics.
- World 2, the social environment, is the world of 'subjective experience', of people and the dynamics of their mental experience.
- World 3, the knowledge environment, is the world of 'statements in themselves', of theories and the dynamics of their logical development.

All the phenomena of expertise acquisition and transfer discussed in the previous section can be represented and analyzed in terms of this diagram. The expert role develops through interaction with the three environments shown: direct experience of the dynamics of the physical, social and knowledge environments; learning by example, evaluation and linguistic communication from other experts in the social environment; managed experience and transmission; and so on. Figure 1 provides a general framework for knowledge acquisition and transfer, including knowledge engineering for expert systems.

However, the diagram only gives expression to the phenomena, making some significant distinctions, but giving no underlying model for the processes involved. The remainder of the paper develops such an overall model of knowledge acquisition and transfer.

# HUMANITY AS A DISTRIBUTED ANTICIPATORY SYSTEM

It is difficult to know where to begin in analyzing human knowledge processes. The theoretical foundations to be developed are based on the concept that humanity can be characterized as a *distributed anticipatory system* —a coordinated collection of autonomous systems whose primary dynamic is the anticipation of the future. The motivation for this concept links back to the processes underlying the survival of the species. To survive, a living system must have access to the necessary resources and be capable of coping with threats to its survival. Systems evolve to maximize their access to resources and minimize their vulnerability to threats.

If the universe were static a simple model of resource availability and prey/predator relations would determine the dynamics of living systems. Until the advent of humanity this planet was a static universe over long periods of evolutionary time. The beginnings of the humanity were set in this static universe but its activities soon began to change that universe so that uncertainty about the future began to dominate our survival processes. Changes in the earth that would have taken millions of years began to occur over millenia, and now they occur within our lifetimes. Humanity developed resources far beyond their natural availability, extinguished most predators other than itself, and changed the ecology of the earth. Much of this planet is now a human construct and the distinction between natural and artificial has become meaningless. Figure 2 shows the system dynamics underlying humanity. The shaded areas show the response to uncertainty. The keys to survival in an uncertain world are three-fold: first to maximize territorial dispersion so that some part of the system is outside the range of a threat; second to maximize cultural diversity so that some part of the system has the capability to survive a threat; third to improve anticipation of the future, passively to predict threats in advance, and actively to rebuild the universe so that they do not occur.



Figure 2 Dynamics of a survival-driven system

Figure 2 encompasses the key technological phenomena of our time. Information technology increases control of information, giving computing when implemented in silicon and genetic engineering when implemented in DNA. The space program aims to achieve territorial dispersion and access to new resources. Genetic engineering aims to achieve cultural diversity and improvement of existing resources. Information technology is being used to model our anticipatory processes, leading to their greater understanding through cognitive science and their enhancement through artificial intelligence.

Characterizing humanity as a distributed anticipatory system follows from the logic of Figure 2. The survival-driven system is the species rather than the individual, and the main knowledge acquisition system is the species. Humanity has achieved territorial dispersion and cultural diversity through its structure as distributed collection of autonomous entities. Language has evolved as a means of coordinating the activities of these autonomous entities. It is essential to many of the processes of knowledge acquisition and transfer. Interaction with the social and knowledge environments is at least as important as interaction with the physical world. The evolutionary pressures have been very strong in selecting genes giving the capability for the species to act as a single distributed individual, combining autonomy and cohesion through enhanced linguistic communication.

# **KNOWLEDGE ACQUISITION AS A MODELING PROCESS**

The characterization of humanity as a distributed anticipatory system, modeling the world so as to increase its probability of survival, places the systemic foundations of knowledge processes in theories of modeling. Klir (1976, 1985) has provided the most general account of modeling, and proposed an epistemological hierarchy accounting for the main components of any modeling systems and their inter-relations. Gaines (1977) gave a mathematical formulation of the general problem of modeling as a relation between order relations at different levels of this hierarchy. The hierarchy has proved a valuable conceptual tool in analyzing a wide variety of modeling systems both in terms of their ontological presuppositions and their epistemological processes.



Figure 3 Epistemological hierarchy of a system modeling a world

Underlying the modeling hierarchy is the notion of a distinction as the primitive concept underlying the representation of knowledge. It is a sufficient primitive to give foundations for systems theory including the notion of a system itself (Gaines 1980). In its psychological form, as a personal construct (Kelly 1955), the notion has been used to derive very effective techniques for knowledge transfer from experts to expert systems (Boose 1985). Its foundational role in knowledge acquisition is evident in the hierarchical representation of distinctions in a modeling system shown in Figure 3. The levels of the hierarchy itself are the results of distinctions that we make so that no additional primitives are introduced—in Klir's (1976) terminology:-

- The source system is distinguished as those distinctions that the particular modeling system makes—it is a distinction about distinctions defining the construct system of an individual.
- The data system is distinguished as those distinctions that have been made about a particular event—a distinction about distinctions defining an event.
- The generative system is distinguished as a set of distinctions that also defines an event—these are model-generated rather than event-generated—it is the match between the model-generated and event-generated distinctions that determines the degree of approximation of the model to the world—this is a distinction about distinctions among distinctions that defines goodness of fit.
- The structure system is distinguished as a set of distinctions that compare models—it is the order relation of simplicity/complexity on models that determines the preference for the simplest model that is an adequate approximation to the world—this is a distinction about distinctions that defines our preference for simple models.
- The meta system is distinguished as a set of distinctions that specify the basis of these comparisons.
- The meta-meta system, and higher levels, are distinguished as sets of distinctions that specify further relations among the distinctions on the level below.

Note that the upper levels of modeling are totally dependent on the system of distinctions used to express experience through the source system.

Distinctions are not just static partitions of experience. They may be operations: actions in psychological terms; processes in computational terms. Whether a system finds a distinction in the world, imposes it passively as a view of the world, or imposes it actively as a change in the world, is irrelevant to the basic modeling theory. It makes no difference to the theory whether distinctions are instantiated through sensation or action. In knowledge engineering we have to incorporate both the expert's prediction and control processes.

# KNOWLEDGE ACQUISITION IN THE HIERARCHY

The hierarchy of Figure 3 accounts for knowledge acquisition as the modeling of events enabling adequate prediction and action. A modeling schema results from distinctions about distinctions at each level in the hierarchy. In prediction the key distinction is to what degree a level accounts for the information flowing through it and hence this distinction may be termed one of *surprise* (Gaines 1977), in the sense used by the economist Shackle (1955). Surprise goes in opposition to the degree of membership of a predicted event to an actual event and the expected surprise is a form of entropy. Surprise at the lowest level of the hierarchy corresponds to distinctions being inadequate to capture events; surprise at the next level to inadequate variety to experience events; at the next level to inadequate approximation to predict events; at the next level to inadequate comprehensiveness to account for events.

The formal theory of modeling is one in which models are selected at each level down the hierarchy to minimize the rate at which surprise is passing up the hierarchy. The criteria for model selection independent of the data are generally thought of as being ones of simplicity/complexity: of two models which fit the data equally well choose the simplest. However, notions of simplicity/complexity are not well-defined nor intrinsic to the class of models. The simplicity/complexity ordering is arbitrary and in its most general form is just one of *preference*. Hence the general modeling schema is one in which surprise flows up the hierarchy and preference flows down. In situations that are mathematically well-defined, such as determining the structure of a stochastic automaton from its behavior, such a model schema gives the correct results (Gaines 1977). Conversely, the success of the schema in stabilizing with regard to a given world defines the characteristics of that world.

Thus the basic modeling schema for learning from experience is one in which surprise flows up the hierarchy and preferences flow down. In primitive organisms only the lower levels of the hierarchy are developed, surprise is generated from experience and preference is genetically encoded. In higher organisms the modeling process generalizes both surprise and preference to cope with novel environments. Humanity has developed the upper levels of the hierarchy and detached surprise from experience and preference from its genetic roots. Surprise can flow up from a level without flowing into it from below because the processes at that level have generated novelty. Preference can be generated at a high level detached from both experience and genetic origins and flow down to affect the relations of the organism to the world.

There is neurological and behavioral evidence of the existence within the brain of the two channels of communication shown in Figure 3 (Tucker & Williamson 1984). The arousal system passes surprise up to the cortex from the limbic region as unexpected events occur. The activation system passes preferences down from the cortex to the motor regions.

### **COGNITIVE SCIENCE INTERPRETATION OF THE HIERARCHY**

The loop in Figure 1 from events through distinctions up through the modeling hierarchy and then down again to predictions and actions corresponds to the epistemological model of man as an anticipatory system developed by Kelly (1955). Thus, the systemic hierarchy of Figure 3 has an analog in psychological terminology as shown in Figure 4:-

- The source level is one of constructs, distinctions made in interacting with the world.
- The data level is one of experiences, events which happen to us, and we make happen, in terms of the distinctions already made.
- The generative level is one of hypotheses which are rationalizations of experience.
- The structure level is one of analogies which are correspondences between these rationalizations.
- The meta level is one of abstractions which are foundations of analogies.
- The meta-meta level is one of transcendencies which are accounts of abstractions.



Figure 4 Construction hierarchy of a person modeling a world

# EXPERTISE, ROLES, GROUPS AND SOCIETIE

The anticipatory processes of the modeling hierarchy may be extended to the operation of society by viewing groups as cross-sections comprising multiple individuals. This concept may be given deeper significance by considering the inductive inference process underlying knowledge acquisition. Whereas deductive logical inference is well-understood and well-founded, the inductive inference that underlies knowledge acquisition is not. Deduction guarantees to take us from valid data to valid inferences, but the inferences are thereby part of the data—no new knowledge is generated. Induction takes us from valid data to models of that data that go beyond it—by predicting data we have not yet observed, and by giving explanations of the data in terms of concepts that are unobservable. Induction generates new knowledge but, as Hume (1739) pointed out over 200 years ago, the process is not deductively valid and it is a circular argument to claim that it is inductively valid.

Philosophers have continued to debate Hume's arguments and search for justification of the inductive process. Goodman (1973) proposed that we accept the circularity but note that it involves a dynamic equilibrium between data and inference rules as shown in Figure 3: "A rule is amended if it yields an inference we are unwilling to accept; an inference is rejected if it violates a rule we are unwilling to amend." Rawls (1971) in his theory of justice terms this a reflective equilibrium. Stich and Nisbett (1984) noted flaws in Goodman's argument and repaired them by proposing that the equilibrium is social not individual: "a rule of inference is justified if it captures the reflective practice not of the person using it but of the appropriate experts in our society." This gives an operational definition of the role of experts within an anticipatory system as referential anticipatory sub-systems coordinating the overall knowledge acquisition process. It also leads to an concept of the role of expert systems in society, as making the referential process more overt and widely available.



Figure 5 Reflective equilibrium in inductive inference

The extension of the modeling hierarchy to social processes is straightforward since Figure 3 presents a general modeling schema and applies as much to groups of people, companies and societies as it does to the roles of a person. The epistemological hierarchy of a person is a cross-section of the epistemological hierarchy of the society generating their life-world. Pask's (1975) concept of P-Individuals as the basic units of psycho-socio-processes allows roles, people, groups, organizations and societies to be treated in a uniform framework. An individual is defined in cognitive terms as a psychological process (Pask 1980) and more complex psychological and social structures may be defined similarly by taking into account the possibilities of timesharing, process switching and distributed processing with psychological processors. For example, one person may assume many psychological roles (process switching), whereas a group of people working together may act as a single goal-seeking entity and hence behave as one process (distributed processing).

# **REPRESENTATION OF EXPERT SKILLS WITHIN THE HIERARCHY**

In the analysis of expertise, the skills to achieve goals in the world are the crucial capabilities of the modeling system, and the usual hierarchical models of skills (Powers 1973, Rasmussen 1983) are naturally subsumed within the modeling formulation. Figure 6 shows the basis for action at different levels in the modeling hierarchy:-

- At level one, the activation of a construct may be linked directly to a primitive act, another construct. This corresponds to reflex actions and stimulus-response connections. In system-theoretic terms this level might be implemented by conditional probability calculations giving confirmation-theoretic inductive inference.
- At level two, constellations of experience may be linked to complete action sequences through rules derived from similar experience. In system-theoretic terms this level might be implemented by fuzzy production rules giving generalization-based inductive inference. These constellations may be regarded as prototypical schema, or frame-like structures in computational terms.
- At level three, a generative model of experience, may be used to compute an optimal action sequence. In system-theoretic terms this level might be implemented by a state-based modeling scheme giving model-based inductive inference.
- At level four, a variety of alternative models may be compared as a basis for selecting one appropriate to the required goals. In system-theoretic terms this level might be implemented by a category-theoretic functional analysis scheme giving analogical inductive inference.
- At level five, generalized abstract models may be used as templets from which to instantiate one appropriate to the required goals. In system-theoretic terms this level might be implemented by a category-theoretic factoring scheme abstracting the mathematical form of an analogy and giving abstractional inductive inference.
- At level six, the entire process described may be transcended through a recognition that it is based on distinctions being made at various level, and an attempt to rationalize these distinctions and create new ones. In system-theoretic terms this level might be implemented by a distinction-based analysis scheme giving what might be termed transcendental inductive inference.



Figure 6 Action hierarchy of system modeling a world

It is an interesting comment on the state of the art in computer science that it has proceeds "middle-outward" in its representation of the knowledge involved at different levels of the hierarchy. Information technology has been primarily concerned with level three activities, and is only now beginning through expert systems to emulate level two activities.

### LANGUAGE AND CULTURE IN KNOWLEDGE ACQUISITION

The creation of new knowledge takes place through the surprise/preference flows within the hierarchy and it is these processes that determine the rate of scientific discovery, technological invention and product innovation. The capability for an entire society to act as a distributed knowledge acquisition system depends on communication processes to coordinatie activity at a given level of the hierarchy across different people. This process whereby each person does not have to undertake all aspects of the inductive process but can share the results of such processing by others supports what is generally termed the culture of a society (Vanderburg 1985). People use language for much of this communication but they also have in common with other animals the capability to communicate cultural information without the use of language. Mimicry is an important mechanism for knowledge acquisition as is reinforcement through reward and punishment.

The human development of language enables coordination to take place in a rich and subtle fashion that greatly enhances, but does not replace, the more basic mechanisms in common with other species. It is particularly important at the upper levels of the hierarchy where direct expression is difficult. From an individual point of view, language is a way of bypassing the normal modeling procedures and interacting directly with the system at any level. In particular it can directly affect the preference system. Much skilled activity is not directly accessible through language, but even when language cannot mediate the direct transmission of knowledge it may be used to achieve the same effect by the indirect support of other mechanisms. For example, one can describe a good learning environment, or a behavior in sufficient detail for mimicry.

Figure 7 shows the cultural support for knowledge acquisition at different levels in the modeling hierarchy:-

- The reflexive knowledge at level one has no verbal component and comes directly from experience, often that of mimicking the behavior of others. This level has been termed informal to correspond to Hall's (1959) definition of cultural transmission of behavior of this type.
- The rule-based knowledge at level two is usually transmitted by reinforcement of behavior, verbal rules, or is induced from the patterns of knowledge at level 1. This level has been termed formal to correspond to Hall's definition of cultural transmission of behavior of this type.
- The computational knowledge at level three is usually transmitted by technical explanation, or is induced from the patterns of knowledge at level two. This level has been termed technical to correspond to Hall's definition of cultural transmission of behavior of this type.
- The comparative knowledge at level four is usually transmitted by simile and metaphorical analysis, or is induced from the patterns of knowledge at level three. Hall does give a name to this level but the term comparative captures his own activity of highlighting the features of one culture by contrasting it with others.
- The abstract knowledge at level five is usually transmitted through mathematical representation, or is induced from the patterns of knowledge at level four.
- The transcendental knowledge at level six is usually transmitted by general systemtheoretic analysis, or is induced from the patterns of knowledge at level five. Many mystical and consciousness-raising techniques may be seen as attempts to communicate knowledge at this level when formal analysis is impossible. It involves moving outside the framework established at the lower levels. Pope (1984) has given examples of this process in a wide range of cultures.



Figure 7 Cultural transmission hierarchy of people modeling a world

# **KNOWLEDGE TRANSFER PROCESSES**

The action and cultural transmission hierachies of Figures 6 and 7 may be combined to give an overall framework for knowledge representation, processing, acquisition and transmission as shown in Figure 8. This gives detailed foundations for the processes of knowledge engineering discussed in the first part of this paper and shown in Figure 1.

The primary learning process is one of direct acquisition based on attempts to solve tasks through interaction with a world, as shown in the lower part of Figure 8. This is applicable to all three worlds: physical, social and knowledge.

This primary process is the only one open to humanity as a species and is distributed across partially autonomous individuals coordinated by cultural communication processes. From an individual perspective it is these communication processes that dominate knowledge acquisition and transfer.

The right hand side of figure 8 shows the cultural communication processes supporting knowledge transfer:-

• At level one mimicry by watching experts is the main transfer mechanism.

- At level two reinforcement by working under expert supervision is the main transfer mechanism.
- At level three rational explanation from interviewing experts and books is the main transfer mechanism.
- At level four use of analogous models to derive knowledge from similar situations is the main transfer mechanism.
- At level five use of basic laws to derive specific models is the main transfer mechanism.
- At level six use of system priciples to derive specific laws is the main transfer mechanism.

Levels 1 and 2 of the cultural transmission process, mimicry and reinforcement, clearly belong to world 2, the social environment of subjective experience. Levels 4 through 6, analogies, laws and principles, clearly belong to world 3, the knowledge environment of statements in themselves. Level 3, of rational explanation, bridges both worlds.



Figure 8 Knowledge transfer processes in 3 worlds

# **KNOWLEDGE TRANSFER METHODOLOGIES**

One practical consequence of the analysis of knowledge acquisition and transfer given above is a hierarchy of knowledge transfer methodologies based on Figure 8 which defines the areas of application of the knowledge engineering techniques already in use. Figure 9 shows this hierarchy in relation to the basic architecture of an expert system:

Knowledge base: the central feature is a knowledge base of facts and inference rules.

- *Expert system shell*: the lower oval shows the operational system for applying the knowledge base to knowledge support.
- The inference system derives the consequences of facts about a particular situation.
- The planning system determines how to use the inference system to satisfy the specified objectives.
- The explanation system answers questions about the basis of the inferences made.
- *Knowledge acquisition system*: the upper oval shows the knowledge transfer processes that may be used to establish the knowledge base.
- The knowledge generation system implements knowledge acquisition by the raw induction of models from experience without cultural support as exemplified by ATOM (Gaines 1977) and AM (Davis & Lenat 1982).
- The expertise modeling system implements knowledge acquisition by mimicking an expert's behavior as exemplified by INDUCE (Michalski & Chilausky 1980).
- The performance reinforcement system implements knowledge acquisition from performance feedback as exemplified by the PERCEPTRON (Rosenblatt 1958) and STelLA (Gaines & Andreae 1966).
- The knowledge elicitation system implements knowledge acquisition by interviewing experts as exemplified by ETS (Boose 1985), KITTEN (Gaines & Shaw 1986), MORE (Kahn, Nowlan & McDermott 1985) and SALT (Marcus, McDermott & Wang 1985).
- The knowledge structuring system implements knowledge acquisition by analogy as exemplified by TEIRESIAS (Davis & Lenat 1982) and CYC (Lenat, Prakash & Shepherd 1986).
- The basic laws system implements knowledge acquisition by building physical models as exemplified by simulation languages such as SIMULA (Nygaard & Dahl 1981).
- The systemic principles system implements knowledge acquisition by derivation from abstract principles as exemplified by the category-theoretic language OBJ (Goguen & Meseguer 1983).

Most current expert system shells offer a limited knowledge base represented in terms of production rules and frames, have weak planning capabilities limited to deriving consequences and testing assertions, and give simple explanations in terms of the facts and rules used. However, rapid improvements are taking place in knowledge representation, inferencing and explanation capabilities of expert systems (Hayes-Roth 1985, Michalski & Winston 1986, Pearl 1986, Swartout 1985).

Most current knowledge acquisition is done by manual knowledge elicitation. However, rapid improvements are taking place in the knowledge automatic knowledge acquisition capabilities of expert systems (Boose 1985, 1986, Bradshaw & Boose 1986, Gaines & Shaw 1986, Kahn, Nowlan & McDermott 1985, Marcus, McDermott & Wang 1985, Shaw & Gaines 1986, van de Brug, Bachant & McDermott 1985).

The knowledge base and shell have been incoprated with the acquisition methodolgies in Figure 9 to show the natural relationships between all three. Shell structure and knowledge acquisition research and development have so far been treated as separate enterprises. Integrating the two is essential to future knowledge-based system applications so that the combined system provides maximal and coherent support to the expert and knowledge engineer.



Figure 9 Knowledge acquisition hierarchy for expert systems
## CONCLUSIONS

The distributed anticipatory system formulation of knowledge acquisition and transfer processes outlined in this paper provides scientific foundations for knowledge engineering:-

- It provides an operational model of the notion of expertise and the role it plays in our society, and hence provides formal foundations for expert systems.
- It provides a detailed structure for the cognitive processes involved in knowledge acquisition and transfer.
- There are links between the model and neurological processes in the brain.
- It suggests that the basic cognitive system that should be considered is a social organization, rather than an individual.
- Computational models of inductive inference already developed can be applied directly to the social model.
- Cultural knowledge acquisition and transmission processes are explicitly identified and structured within the model.
- Linguistic communication coordinates the distributed autonomous units forming the anticipatory system.
- It suggests that groups of experts, rather than individuals, should be used in developing expert systems.
- It provides a framework within which all the methodologies for knowledge acquisition and transfer currently under developed can be incorporated, analyzed and compared.

There are several implications for future developments in Figures 8 and 9. Whereas past knowledge acquisition and transfer systems have been developed using techniques at one level of the hierarchy, future systems will increasingly follow the example of human apprenticeship and operate at multiple levels. The combination of teaching by example, performance feedback and explicit statement is ubiquitous in human knowledge transmission and will become increasingly so in expertise transfer systems. The combination of rule-based, model-based, analogical and law-based inference is ubiquitous in human knowledge processing and will become increasingly so in expert systems.

The range and variety of expertise transfer systems being developed is very important. They are complementary rather than competetive. We are entering a phase of system integration when the best features of many systems will be melded to provide a new generation of knowledge acquisition and transfer systems. This workshop will give a major impetus to the collaboration necessary to the development of this next generation of systems, and hence to the progress and application of the major new information technology of knowledge-based computing systems.

## ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the National Sciences and Engineering Research Council of Canada.

## REFERENCES

Bainbridge, L. (1979). Verbal reports as evidence of the process operator's knowledge. International Journal of Man-Machine Studies, 11(4), 411-436 (July).

- Bainbridge, L. (1986). Asking questions and accessing knowledge. Future Computing Systems 1, in press.
- Boose, J.H. (1985). A knowledge acquisition program for expert systems based on personal construct psychology. International Journal of Man-Machine Studies 20(1), 21-43 (January).
- Boose, J.H. (1986). Rapid acquisition and combination of knowledge from multiple experts in the same domain. Future Computing Systems, In press.

- Bradshaw, J.M. & Boose, J.H. (1986). NeoETS. Proceedings of North American Personal Construct Network Second Biennial Conference. pp. 27-41. University of Calgary: Department of Computer Science (June).
- Broadbent, D.E., FitzGerald, P. & Broadbent, M.H.P. (1986). Implicit and explicit knowledge in the control of complex systems. British Journal of Psychology 77, 33-50.
- Collins, H.M. (1985). Changing Order: Replication and Induction in Scientific Practice. London: SAGE.
- Davis, R. & Lenat, D.B. (1982). Knowledge-Based Systems in Artificial Intelligence. New York: McGraw-Hill.
- Dixon, N. (1981). Preconscious Processing. Chichester: Wiley.

Freud, S. (1914). Psychopathology of Everyday Life. London: Benn.

- Gaines, B.R. (1977). System identification, approximation and complexity. International Journal of General Systems, 3, 145-174.
- Gaines, B.R. (1980). General systems research: quo vadis ?. Gaines, B.R., Ed. General Systems 1979. Vol. 24, pp. 1-9. Kentucky: Society for General Systems Research.
- Gaines, B.R. & Andreae, J.H. (1966). A learning machine in the context of the general control problem. Proceedings of the 3rd Congress of the International Federation for Automatic Control. London: Butterworths.
- Gaines, B.R. & Shaw, M.L.G. (1986). Knowledge engineering techniques. Proceedings of AUTOFACT'86. To appear. Detroit (November).
- Goguen, J.A. & Meseguer, J. (1983). Programming with parametrized abstract objects in OBJ. Theory and Practice of Programming Technology. Amsterdam: North-Holland.

Goodman, N. (1973). Fact, Fiction and Forecast. Indianapolis: Bobbs-Merrill.

Hall, E.T. (1959). The Silent Language. New York: Doubleday.

Hawkins, D. (1983). An analysis of expert thinking. International Journal of Man-Machine Studies, 18(1), 1-47 (January).

Hayes-Roth, B. (1985). A blackboard architecture for control. Artificial Intelligence, 26(3), 251-321 (July).

Hayes-Roth, F., Waterman, D.A. & Lenat, D.B., Eds. (1983). Building Expert Systems. Reading, Massachusetts: Addison-Wesley.

Hume, D. (1739). A Treatise of Human Nature. London: John Noon.

Kahn, G., Nowlan, S. & McDermott, J. (1985). MORE: an intelligent knowledge acquisition tool. Proceedings of the Ninth International Joint Conference on Artificial Intelligence. pp. 581-584. California: Morgan Kaufmann.

Kelly, G.A. (1955). The Psychology of Personal Constructs. New York: Norton.

- Klir, G.J. (1976). Identification of generative structures in empirical data. International Journal of General Systems, 3, 89-104.
- Klir, G.J. (1985). Architecture of Systems Problem Solving, New York: Plenum Press.
- Lenat, D., Prakash, M. & Shepherd, M. (1986). CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. AI Magazine 6(4), 65-85.
- Marcus, S., McDermott, J. & Wang, T. (1985). Knowledge acquisition for constructive systems. Proceedings of the Ninth International Joint Conference on Artificial Intelligence. pp. 637-639. California: Morgan Kaufmann.
- Michalski, R.S. & Chilausky, R.L. (1980). Knowledge acquisition by encoding expert rules versus computer induction from examples—A case study involving soyabean pathology. International Journal of Man-Machine Studies, 12, 63-87.
- Michalski, R.S. & Winston, P.H. (1985). Variable precision logic. Artificial Intelligence, 29(2), 121-146 (August).
- Nisbett, R.E. & Wilson, T.D.. (1977). Telling more than we can know: verbal reports on mental processes. Psychological Review 84, 231-259.

Norman, D. (1980). Twelve issues for cognitive science. Cognitive Science, 4(1), 1-32.

Nygaard, K. & Dahl, O-J. (1980). The development of the SIMULA languages. Wexelblat, R.L. (Ed.) (1981). History of Programming Languages. pp. 439-480. New York: Academic Press.

- Pask, G. (1975). Conversation, Cognition and Learning. Amsterdam: Elsevier.
- Pask, G. (1980). Developments in conversation theory—Part I. International Journal of Man-Machine Studies, 13(4), 357-411 (November).
- Pearl, J. (1985). Fusion, propagation and structuring in belief networks. Artificial Intelligence, 29(3), 241-288 (September).

Pope, S. (1984). Conceptual synthesis: beating at the ivory gate?. Smith, W., Ed. Systems Methodologies and Isomorphies. pp. 31-40. California: Intersystems.

Popper, K.R. (1968). Epistemology without a knowing subject. Van Rootselaar, B., Ed. Logic, Methodology and Philosophy of Science III. pp. 333-373. Amsterdam, Holland: North-Holland Publishing Co..

Powers, W.T. (1973). Behavior: The Control of Perception. New York: Aldine.

- Rasmussen, J. (1983). Skills, rules and knowledge; signals, signs and symbols, and other distinctions in human performance models. IEEE Transactions on Systems, Man & Cybernetics, SMC-13(3), 257-266 (May/June).
- Rawls, J. (1971). A Theory of Justice. Cambridge, Massachusetts: Harvard University Press.
- Rogers, C.R. (1967). On Becoming a Person: A Therapist's View of Psychotherapy. London: Constable.

Rosenblatt, F. (1958). The Perceptron: a probabilistic model for information storage and organization in the brain. Psychological Review, 65, 386-407 (November).

Shackle, G.L.S. (1955). Uncertainty in Economics. Cambridge University Press.

Shaw, M.L.G. & Gaines, B.R. (1986). Techniques for knowledge acquisition and transfer. Proceedings of AAAI Workshop on Knowledge Acquisition for Knowledge-Based Systems. Banff (November).

- Stich, S.P. & Nisbett, R.E. (1984). Expertise, justification and the psychology of inductive reasoning. Haskell, T.L., Ed. The Authority of Experts. pp. 226-241. Bloomington, Indiana: Indiana University Press.
- Swartout, W. R. (1985). Knowledge needed for expert system explanation. Proceedings of the 1985 National Computer Conference. Vol. 54, pp. 93-98. Reston, Virginia: AFIPS Press.

Tucker, D.M. & Williamson, P.A. (1984). Asymmetric neural control systems in human self-regulation. Psychological Review, 91(2), 185-215 (April).

van de Brug, A, Bachant, J. & McDermott, J. (1985). Doing R1 with style. Proceedings of the Second Conference on Artificial Intelligence Applications. IEEE 85CH2215-2, pp. 244-249. Washington: IEEE Computer Society Press.

Vanderburg, W.H. (1985). The Growth of Minds and Cultures: A Unified Theory of the Structure of Human Experience. Toronto: University Press.

Welbank, M. (1983). A Review of Knowledge Acquisition Techniques for Expert Systems. BTRL, Ipswich: Martlesham Consultancy Services.

Shortliffe, E.H. (1976). Computer-Based Medical Consultations: MYCIN. New York: Elsevier.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# Knowledge-Based Knowledge Acquisition for a Statistical Consulting System

William A. Gale AT&T Bell Laboratories 2C278 600 Mountain Avenue Murray Hill, NJ, 07974

## ABSTRACT

Knowledge-based knowledge acquisition means restricting the domain of knowledge that can be acquired and developing a conceptual model of the domain. We have built a prototype knowledge-based knowledge acquisition system for the domain of data analysis. A critique of the prototype has led to a design for a possibly practical data analysis knowledge acquisition system.

## 1. Introduction

Daryl Pregibon and I built our first statistical consultation system, REX, using standard expert system techniques (Gale, 1986a; Pregibon and Gale, 1984). REX is a consultation program in regression analysis, a statistical technique for data analysis. It demonstrated the feasibility of using expert system techniques to construct data analysis consultation systems. It had an active life as a demonstration system, running weekly about for a year.

We then faced the problem of building consultation systems for other analytic techniques, which brought home the inadequacies of the knowledge acquisition methods used in building REX. The next (second) section reviews the shortcomings of standard expert system construction methods as we experienced them in a statistical application. Similar shortcomings have been noted in many application areas, leading to a widespread perception of need for better knowledge acquisition methods.

REX made two major contributions to subsequent work. The first is a viewpoint for thinking about data analysis as a diagnostic problem. Briefly, one should list model assumptions (analogous to possible diseases), test the data set at hand for violations of the assumptions (analogous to symptoms), and if found select a transform of the data (analogous to treatment). The success of this approach depends on the representation of statistical knowledge. This is the second major contribution of REX. REX has a set of statistical primitives including tests, plots, assumptions, and transforms, which can be implemented as frames with slots containing procedures, or as objects (classes) with instance variables and methods. The hierarchical structure of the network of frames directs the interpretation of the statistical knowledge.

We refer to the statistical knowledge as *strategy*. A hand held calculator can do (some) regression *calculations*. But the assumptions behind this calculation, how they can be examined, when they should be examined, and how they can be remedied if found

require considerably more expressive power than even such elaborate calculators as modern statistical packages.

Student was subsequently designed to allow expert statisticians working alone to build and test strategies (Gale, 1986b). It was conceived of as a program by example system, because of the key role that examples seemed to play in the construction of REX. We built a prototype that acquired knowledge from a *first* example in a new data analysis area. The third section describes the knowledge acquisition techniques used in the prototype, and the fourth criticizes them.

One successful part of the Student prototype is knowledge-based knowledge acquisition. The key to this technique is restricting the domain for which knowledge can be acquired. Just as domain restriction allows domain specific knowledge to be used in a knowledge-based consultation system, domain restriction allows domain specific knowledge to be used in a knowledge-based knowledge acquisition system. The domain specific knowledge essential for knowledge-based knowledge acquisition is a conceptual framework for the domain. The framework must specify the types of primitive entities to be used, how each primitive will be represented, acquired, and modified, and how the primitives can be combined and displayed. The fifth section discusses the issues that must be resolved in order to use knowledge-based knowledge acquisition.

At the risk of redundancy, let me define knowledge-based knowledge acquisition as the approach to knowledge acquisition which (1) restricts the domain of knowledge which can be acquired and (2) builds a conceptual model of the restricted domain.

The sixth section discusses the generality of knowledge-based knowledge acquisition. It is a practical tool for widening the knowledge acquisition bottleneck when several similar consultation systems must be built.

#### 2. A Critique of Knowledge Acquisition in REX

Developing a strategy for use in REX was a labor-intensive process. Two phases can be distinguished. In the first phase the statistician responsible for the strategy, Daryl Pregibon, chose a half dozen regression examples that clearly showed some frequent problems. He then analyzed them using interactive statistical software with an automatic trace. After analyzing the group of examples, he studied the traces and abstracted a description of what he was doing. We coded this as a strategy for REX and tried it on a few more examples. He revised the strategy completely at this point, and the second phase began.

In the second and longer phase, one of us would select one additional regression example and run REX interactively on the chosen example. Since we selected the example knowing what would stretch REX, REX usually reported a severe problem that it didn't know how to fix. Then we would modify the strategy so that the example would be handled. This process was iterated through about three dozen more examples.

Based on this experience, and on a feeling that it was typical of other techniques, we do not believe it is possible to construct a data analysis strategy without working through many examples. One must make many decisions to construct a strategy, and there is no literature simplifying the task. Therefore the only available defense of a strategy is to demonstrate performance, which requires working many examples more than those used to construct the system. On the other hand, our experience also leads us to believe that it is easy to generalize from data analysis examples. The basis for generalization is usually a statistical test which statisticians can provide. Generalization then consists of determining the range of values of the test for which the demonstrated technique holds.

However, the way in which we worked examples for REX was far from ideal. The first difficulty with our method was assuring ourselves that a strategy modified to work one additional example still worked all previous examples. We could by brute force run REX in batch mode on all previous examples and see if the performance was the same. Usually we reasoned that most of the previous examples could not be affected, and checked the few that might be affected by hand. Naturally, the more examples worked, the more severe this problem became. The need to check consistency in batch mode for a system designed to be interactive reduced the flexibility of the strategy developed.

Second, the method used was the epitome of the currently standard two-person development of expert systems. I built the inference engine used while Daryl was responsible for the strategy developed. Whenever Daryl wanted to do something he hadn't done before, we had to huddle, as Daryl was learning a language he would only use to construct one program. In a department with twenty professional statisticians and one person intimately familiar with the inference engine, it was not clear how many additional data analysis techniques could be handled by this two person approach.

Third, it would be difficult to modify the strategy in REX. Modifiability is important because a growing literature on strategy (Gale, 1986c; Haux, 1986; Darius, 1986) can be expected to suggest desirable changes. It is also important because users will probably want to modify strategies to their particular needs. Statistics is a discipline that is applied in other, "ground" domains. Current knowledge representation and language generation techniques are not adequate to produce a tool that will speak physics with physicists and psychology with psychologists. An alternative to one broad tool is a tool that is readily specialized. However, the first two problems would make this difficult: to specialize the program a local statistician would have to learn a language used by no other program in the world, and the modifications made might inadvertently destroy some capabilities of the strategy.

The contribution of REX to subsequent work was stated in the introduction. It provided us with the beginnings of a conceptual framework for data analysis: a data analysis consists of a desired calculation, assumptions required for the calculation to be meaningful, tests for the violation of the assumptions, and transformations to ameliorate the violations. The classes of frames used in REX provided us with an initial list of classes of primitives that has remained useful and has been expanded.

## 3. Knowledge Acquisition in a Student Prototype

The necessity of working examples to construct a data analysis strategy suggested the possibility of acquiring strategies directly through that process. A system should assist the teacher in establishing consistency across all examples worked, and should not force a statistician to learn an obscure language. It appeared that examples might provide a

language suitable for communication between statisticians and computers.

The first issue encountered in designing Student was how to learn from the *first* example. In a system without knowledge, there is simply no basis for use of information provided in working an example. By providing Student with the conceptual framework induced from REX, we have built a prototype that can deal meaningfully with an example even when it has seen no previous examples. This step was only possible because the system was limited to data analysis.

DeJong's (1983) method of explanatory schema acquisition is another example of the possibility of one-shot learning that is possible with domain knowledge. His approach has more detailed and explicit domain knowledge then ours, and learns independently of human assistance. His technique has not been used for knowledge acquisition for a consulting system, but might be extended that way.

The prototype was implemented on a Symbolics 3670 communicating with a statistics program on a Vax 750.

The conceptual framework used in the Student prototype had the ten classes of primitives shown in the following table. Each instance of a primitive was represented by a frame. In the table, indentation indicates that names of instances of the primitive indented occurred as values in some slot of the superordinate primitive. That is, the relation shown by indentation is "uses information from."

input variables		
data types		
assumption testing		
plot		
	generic	plot
test		
	generic	test

action

question discriminator predicate discriminator

Each primitive has a set of slots, which are also chosen to reflect the structure of data analysis. As an example, perhaps the simplest primitive is the input variable frame, which had only a few slots:

#### input variable

external name of input required or optional default if optional data type variable name

The content of the instances of these primitives is the information that a consultation system must have. For instance, when asking a consulting client for a specific input, it

is necessary to know the common name of the input. Likewise, the system must know whether to insist upon having a given input variable before beginning the analysis (required or optional), and what default to use if the user does not have one of the optional input. The system must also know what data type is required for the input in order to determine if submitted data is possible. Since we do not want to overwrite input data with later calculations, we need a standard variable name to copy the input to.

Gale (1986b) and (1986c) discuss the formalization of data analysis used in Student in more detail. Here it is sufficient to note that the work stands as an example of how the conceptual framework from one consultation system (REX) can be generalized and formalized.

Knowledge-based knowledge acquisition in this context means specifying how the contents of each slot will be acquired. In the case of the input variable primitive, each slot could be acquired by asking the teaching statistician. Most of them could also be acquired more actively. The internal name could be created from the external name and perhaps a unique number. Acceptable data types could be inferred from the data types of the inputs to the set of examples provided. Optional variables and their defaults could be inferred as those with repeated inputs. In fact, it seemed better in each of these cases to ask the teaching statistician and then use the information to check inputs to teaching examples.

Thus, subsidiary techniques designed for the specific knowledge will be chosen. The Student prototype used three subsidiary techniques: interviewing, limits induction, and monte carlo learning.

The preponderance of cases were handled by interviewing. Knowing what is needed, and having a statistician at hand, it is easy to just ask. Even so, exactly how to ask for the information varied between menus, fill in the blank, multiple simultaneous choice, and free response. And of course the prompts varied for each item.

Monte carlo learning was used to establish initial notions of the distributions for test results. The distributions were used in turn to set initial cut points, or *limits* for distinguishing severe, mild and insignificant cases of assumption violations.

Limits induction is inference of limits on test ranges from test results and action (transform) or non-action by the statistician. Let  $v_i$  be the value of a test on the *i*th data set, and  $a_i$  be T or F as the statistician acted or didn't act in the situation covered by the test. Then set the lower cut point as  $\max(v_i | a_i = F)$  and the upper cut point as  $\min(v_i | a_i = T)$ . Then for test values above the upper cut point, the statistician has always acted, and for values below the lower cut point, the statistician has never acted. This simple scheme is slightly modified to include the monte carlo results. Limits induction was programmed in the prototype, but with only single examples, it was not tested.

Knowledge-based knowledge acquisition has several advantages. First, the information in each slot is necessary for a consultation program. Systematizing the knowledge to acquire from a statistician speeds construction because the system won't forget what is needed. Another advantage of knowledge-based knowledge acquisition can be shown in the acquisition of an input variable. It is almost always appropriate to run a number of tests on each input variable by itself. Without knowledge-based knowledge acquisition each time a new variable is given, a battery of tests must be specified by the teaching statistician. However, it is easy to keep track of what tests have been used for all input variables by data type, and to suggest these to the statistician. Since the tests are based only on knowing the data type of the input, they will often be appropriate in many different data analysis procedures. The domain knowledge we are using here is that the tests are similar in many different analysis types, and that they are reasonably organized by data type.

As another example, a statistician may notice after some time of programming that an optional input variable is possible. One would then back up and increase the generality of numerical procedures to accommodate the extra variable. With knowledge-based knowledge acquisition, the statistician is encouraged to think of optional inputs at the beginning of the construction process, thus avoiding the costs of reprogramming. This encouragement may not be effective in all cases, but it can only work in the direction of reducing the problem. In short, by providing a framework for data analysis, the statistician is encouraged to think in previously successful terms.

A system that acquires first examples does not address all the problems in building a knowledge acquisition system. However, the domain restriction is expected to be useful for extending a given body of knowledge as well as initiating it. Extension of knowledge for a given data analytic technique involves demonstrating more assumptions, how to detect their violation, and how to fix them. The same techniques used for initial acquisition suffice here. However, it is also necessary to check consistency for previously worked examples.

Consistency means that after incorporating information on a new assumption, the recommended analyses of all previously worked examples are not changed. This is a requirement analogous to logical monotonicity. Some changes can be proved consistent by using domain knowledge. The domain knowledge consists of a theorem, and the proof consists of verifying the hypotheses of the theorem, so this is not automatic theorem proving. The proof may use data that could be specified and collected when the previous examples were demonstrated. This will be more efficient than rerunning examples. Other cases, such as showing that a new test is not passed for an old example, require new calculations. Domain knowledge may be able to specify data to save which will make such checking faster than completely reworking an example.

Of course, the check may find that a change is inconsistent. That is, that the recommended analysis for at least one previous example has changed. Then the statistician will need to *revise* the existing body of knowledge. This might just consist of blessing the revised analysis for the inconsistent examples. Or it may require revising the strategy, perhaps revising the assumption just added. This can be assisted by domain knowledge encoded as editing procedures.

#### 4. A Critique of Knowledge Acquisition in Student

Interviewing is useful. A knowledge-based interview is easy to write, since one knows exactly what to acquire. Interview procedures attached to slots are easy to keep track of, so that it is easy to see if all slots can be acquired.

A research issue is how much can and should be acquired by interviewing, and how much must or should be provided as initial knowledge. The prototype tested this by attempting to acquire everything by interviewing. It appeared that everything *could* be acquired this way. However, experience with this extreme approach led to deciding to provide some items as initial knowledge. The collected reasons used to justify initial provision of an item were

- (1) distractingly frequent requests for information,
- (2) need for richly structured information,
- (3) need for careful control of the generality of information, and
- (4) stable and non-controversial information.

For example, data types (vectors, matrices, time series, ...) will be built in for reasons 2 and 4. An initial core of technical definitions will be provided for reasons 1 and 4. The domains of functions (so that no attempt will be made to take logs of negative numbers) will be built in for reasons 3 and 4. Reason 4 is cited in each case and appears to be necessary.

Programming by example is possible, but slow and clumsy. There is, however, key information in the examples. And, as argued, examples are necessary. It is useful when describing a plot or test to have an example to do the operations on immediately.

The lisp machine was a successful prototyping environment. However, our consultation systems must use a statistical package for their specialized computations. Therefore, a practical system will probably have to be built on the same machine as the package for reliability of delivery.

The Student prototype assumption testing frames need to be generalized. First, arguments are needed. The prototype was built to redo REX so it copied the REX organization of assumptions. REX had explicitly represented "checking the dependent variable for outliers," and "checking the independent variable for outliers." Thus the Student prototype is only prepared to learn about ground level assumption violations such as "dependent-outlier" and "independent-outlier." Clearly, acquiring a variable-containing entity to represent "checking the input variable for outliers" would be more powerful. Second, the assumption-testing notion needed to be generalized to the notion of "feature," which might never lead to a transformation, but which might lead to a report item.

We found in building REX that the most powerful explanations in statistics were not verbal, but graphical. Thus we programmed before and after plots for each transformation. Student is able to construct these automatically from plots acquired in the course of being shown how to detect an assumption violation. This is a convenience. Generic plots may be another example of knowledge best provided initially. But it may be best to include the capability to acquire them even if it is rarely used.

15-6

Monte carlo learning seems like a technique with much wider applicability for statistical systems to learn about statistical tests. Limits induction is apparently a useful idea although it has not been tested. It can describe what a statistician has actually done, possibly pointing out a poorly worked example, or a poor test. It can be used to alert statisticians to taking an action that is not consistent with previous actions, but can be changed easily if they insist.

#### 5. Issues for Knowledge-Based Knowledge Acquisition

Does the restricted domain have a useful generality? Restriction to the point that one can provide a framework necessarily sacrifices generality. If only one expert system can be made within the restrictions, building a special tool to assist in its construction will be unproductive. Snee (1980) reports a survey result that a dozen data analysis techniques will cover the bulk of data analyzed. So perhaps this many systems would be needed for standard forms of analysis. There are also, however, many specialized forms, practiced in only one industry or one company. After building S (Becker and Chambers, 1984), it was observed that S was being used frequently to construct specialized analytic environments. This suggests a need for many statistical expert systems.

## What is the conceptual framework for the domain? Specifically...

What classes of primitives are needed? Probably the most efficient way to develop a conceptual framework is to build one expert system in the domain. The classes of primitives used are highlighted if one uses a frame based programming approach. The different classes of primitives have different slots and different control procedures. REX was programmed using a system allowing either rules or frames. It turned out that the frames were more useful. A side effect was to identify the classes of primitives used. The Student prototype was built to use frames, and thinking in terms of the classes of primitives made its structure clearer from the start.

How are the primitives structured? Slots in frames which refer to other frames induce a network structure on a collection of frames. This structuring is easy to implement, and has been sufficient for representing assumptions using plots, tests, and transforms in slots. It has not been sufficient for representing a strategy using assumptions, because control concepts (branch and loop) turned out to be necessary. But the additional structure was easy to add.

How is the knowledge validated? This will be highly dependent on the domain. Validation needs to be supported as much as possible, however. In our case, the knowledge is validated by the coincidence of machine generated analyses of specific examples and expert analyses of the same examples. Therefore Student is designed to collect and display examples and their analyses. It can then check the required coincidence automatically.

What background knowledge will be supplied? This will also depend strongly on the domain. Knowledge at the periphery of the domain, such as knowledge of mathematics as required for statistics, is likely to have the required stability and to be non-controversial. The background knowledge identified in our case consists of knowledge

about data types, functions, plot types, and vocabulary.

How is the knowledge communicated? We are trying to build systems that will allow subject matter experts to construct a consultation system. The conceptual framework must be communicated clearly to the expert using the system. Indeed, the expert must be actively encouraged to think in the specific terms provided by the framework, even if it is a natural one.

Probably the most important area to communicate is the structuring of primitives. We have found graphical displays of the hierarchy to be useful here.

### 6. The Generality of Knowledge-Based Knowledge Acquisition

The issues raised in the previous section lead directly to an assessment of the generality of knowledge-based knowledge acquisition.

First, it is clear that it applies to areas where more than one expert system is to be built. It will, in fact, usually be necessary to build at least one expert system manually before it is clear what a conceptual framework for a domain might be. My impression is that many expert systems built to date are representatives of similar ones that might also be built. For instance, medical diagnosis and mineral deposit detection have many specific diseases and minerals to deal with. Thus there may be many domains with both sufficient generality and sufficient restriction.

Second, a conceptual framework is necessary. In building a first, ground level, system it will help to seek regularity and common cases. A frame based programming system helps to identify these commonalities. But, if the first system has a myriad of types of rules or frames, a generalized framework is unlikely.

Third, the framework must be readily presentable. The subject matter specialists may need to be encouraged to think within the specific framework provided, even if it is natural. For hierarchically organized knowledge, graphs are an attractive communication medium. Some restriction in expressiveness may be appropriate to get easier communication.

### 7. Summary

Knowledge-based knowledge acquisition is possible when the domain of knowledge acquisition can be restricted and a conceptual framework for the restricted domain provided. It is useful when the restricted domain still allows constructing a variety of expert systems. We have demonstrated the feasibility of using knowledge-based knowledge acquisition in one domain, data analysis, by building a prototype. The method should be usable in other domains. The principal requirements on the domain are (1) that several independent consulting systems are needed, and (2) that the conceptual framework from one consultation system can be formalized.

#### References

Becker, R. A., and J. M. Chambers, 1984, S: an Interactive Environment for Data Analysis and Graphics, Wadsworth, Belmont, California.

Darius, P., 1986, "Building Expert Systems with the Help of Existing Statistical Software: An Example", *Proceedings COMPSTAT86*, Physica-Verlag, Vienna, Austria, p. 277.

DeJong, G., 1983, "Acquiring Schemata Through Understanding and Generalizing Plans", *Proceedings Eighth IJCAI*, Morgan Kauffman, Los Altos, California, p. 462.

Gale, W. A., 1986a, "REX Review" in W. A. Gale, ed., Artificial Intelligence and Statistics, Addison-Wesley, Reading, Massachusetts, Chapter 9.

Gale, W. A., 1986b, "Student Phase 1: A Report on Work in Progress" in W. A. Gale, ed., *Artificial Intelligence and Statistics*, Addison-Wesley, Reading, Massachusetts, Chapter 11.

Gale, W. A., 1986c, "A Comparison of Representations for Statistical Strategies", *Proceedings of Statistical Computation Section*, ASA, Washington, D.C.

Pregibon, D., and W. A. Gale, 1984, "REX: An Expert System for Regression Analysis" in *Proceedings COMPSTAT84*, Physica-Verlag, Vienna, Austria, p. 242

Haux, R., ed., 1986, Expert Systems in Statistics, Gustav Fischer Verlag, Stuttgart.

Snee, R. D., 1980, "Preparing Statisticians for Careers in Industry" The American Statistician, May 1980, v. 34 pp. 65-75.

## A Conceptual Framework for Knowledge Elicitation

Chaya Garg-Janardan and Gavriel Salvendy<sup>\*</sup> School of Industrial Engineering Purdue University West Lafayette, Indiana 47907.

## Abstract

Knowledge elicitation is a first but critical step in the building of expert systems. The performance of expert systems and the expense involved in their development is determined largely by the accuracy and reliability of the extracted knowledge and the time taken to extract the knowledge, respectively. Hence, the development of a structured, reliable and valid method for knowledge extraction will facilitate the widespread use of expert systems by improving their performance and rendering the building process more economical. This paper presents a conceptual framework that may be used to derive a knowledge elicitation methodology. This conceptual framework is established by extending Newell's and Simon's (1972) problem space concept and integrating it with Kelly's theory of personal constructs.

## 1.0 Knowledge Elicitation

Knowledge elicitation is the process by which facts, rules, patterns, heuristics and operations used by humans to solve problems in the particular domain are elicited. It is a first but critical step in the construction of expert systems. The time required by this phase affects the cost effective development of expert systems. The performance of the expert system in terms of the systems reliability, validity and utility depends on the reliability, validity and accuracy of the elicited knowledge. It is thus of great concern that knowledge elicitation poses a significant bottleneck to the process of building expert systems. Besides humans, knowledge may be elicited from several sources including books, journals, reports, manuals, databases and case studies. However, knowledge elicitation from humans is the major thrust here since the intuition, experience and heuristics used by humans in problem solving are rarely stated explicitly in the literature.

#### 1.1 Important Issues in Knowledge Extraction

A concise and formal statement of the problem of knowledge elicitation precludes the possibility of making explicit the many diverse but interacting factors that contribute to it. To this effect, the statement of the problem is preceded by an examination of issues identified as important to this problem. These include: (a) What knowledge should be elicited, and (b) How should it be elicited, i.e., what attributes should a knowledge elicitation methodology possess? Each of these issues is discussed below.

What knowledge to elicit? Two kinds of knowledge: process and content, may be elicited from humans. Process knowledge is defined as the strategies and

<sup>\*</sup> The writing of this paper and development of the concepts herein were made possible due to fellowships to the first author by NEC Corporation and IBM Corporation and by the NEC Corporation professorship to the second author.

procedures used in problem solving. Content knowledge represents the actual facts and rules used by the human in solving problems. The two kinds of knowledge cannot be divided into mutually exclusive classes. Process knowledge includes the methods by which a subset of the content knowledge is accessed, combined and used to solve problems. Combinations of content and process knowledge and heuristics maybe used successfully so often that they become automated and stored as chunks by the human. As soon as the human recognizes a pattern in a given problem the associated chunk is executed. These chunks form as an individual's experience and expertise level increases and render the process-content distinction more ambiguous. In the context of knowledge extraction, it may be impossible for the knowledge engineer to assert that only one of process or content knowledge will be elicited. This is because, as stated above, the two kinds of knowledge cannot be divided into mutually exclusive classes. Despite this ambiguity, definition of the kind of knowledge that is of primary interest may provide a framework within which to elicit knowledge, evaluate the completeness of the elicited knowledge and prevent the development of methods that elicit random samples of domain specific knowledge. The kind of knowledge that is of primary interest may vary from domain to domain. Elicitation of content knowledge may be the chief concern in analysis type domains; whereas, in synthesis type domains elicitation of both process and content knowledge may be equally important. Analysis problems are those where all possible solutions can be enumerated ahead of time. Synthesis problems are those where unique solutions may be built from components or inputs. In synthesis problems it is not possible to enumerate all possible solutions, at the very outset.

Attributes of a knowledge elicitation methodology. A first step in the formulation of an effective and efficient knowledge elicitation methodology is the enumeration of it's attributes (both required and desirable). The methodology should rest on a theoretical framework, i.e., the procedure used by the methodology should derive from this theoretical framework. A valid theoretical framework facilitates delineation of what knowledge to elicit, provides a bases against which to validate the methodology and the elicited knowledge and structures the procedure used by the method. The methodology should be such that it almost ferrets information out of the human. This is because humans have difficulty in expressing the actual information and strategies used in problem solving. A methodology is needed which draws attention to various aspects of the problem and prompts expression. Instead of eliciting some random subset of knowledge, the methodology should attempt to elicit deep, causal, strategic and nearly complete knowledge. The methodology should not employ procedures that are task specific and vary widely from domain to domain and researcher to researcher. Instead procedures used should be applicable to a class of problem domains.

A knowledge elicitation tool should objectively check for *inconsistencies and* conflicts in the elicited data. Such checks are necessary because humans in their perceptual, response and decision making process are subject to several sources of bias and inaccuracy (Tversky and Kahneman, 1974; Moray, 1985). Checks for conflicts are particularly necessary when eliciting expertise from multiple sources. Expertise Transfer System's (ETS, [Boose, 1986]) method of presenting knowledge in multiple forms to the expert facilitates the detection of conflicts but does not guarantee it. The tool should have a system for combining information elicited from multiple sources or from a single source. ETS incorporates a method to achieve

Appropriate quantitative and qualitative methods should be used to analyze the elicited data, so that any implicit relations, trends and patterns in the elicited knowledge are made explicit. A first step towards the use of quantitative techniques in analyzing data is ENTAIL (Gaines and Shaw, 1981; Boose, 1986). Graesser and Clark (1985) suggest the identification of conceptual graph structures in the elicited information as an effective way of imposing structure and making explicit much of the implicit information. Such analyses will facilitate outputting the elicited data in a structured format. Any hidden relations identified may be used to chunk and establish links in network and frame based representations. The tool should provide facilities using which the elicited data may be examined and modified by the expert. Systems which currently provide this facility include TEIRESIAS (Davis, 1979), ETS (Boose, 1986) and XPLAIN (Swartout, 1984). The format in which the subject responds should be *flexible*. There should be a trade off between flexibility offered and running the risk of permitting the human to ramble on as in undirected 'think aloud' protocol analysis. The individual may be given the option of choosing one of several response formats.

In essence the problem is twofold: to outline a theoretical framework and develop and validate a tool for knowledge acquisition. Both problems need to be solved in light of the above specifications regarding *what* knowledge to elicit, and *how* to elicit this knowledge. This paper includes a solution to the first part of the problem. A solution to the latter part is currently being developed and validated. Prior to outlining a solution to the former part, a brief discussion on the existing methodologies for knowledge elicitation is included.

#### 2.0 Current Methodologies For Knowledge Elicitation

#### 2.1 Protocol Analysis and Related Methods

Protocol analysis is the collection of information from subjects by having them 'think aloud' or introspect and verbalize. Nisbett and Wilson (1977) caution researchers about the pitfalls of protocol analysis. Removal in time, a priori theories (correctly or incorrectly formed), mechanics of judgment, context, nonevents and discrepancy between the magnitude of cause and effect are listed by Nisbett and Wilson as the factors which affect the availability and representativeness of events and stimuli in individuals. These in turn affect the consistency and accuracy of the verbal reports that emerge.

Ericcson and Simon (1984) provide a critical but comprehensive discussion on methods used to collect and analyze protocols and factors that users of protocol analysis should be aware of. Ericcson and Simon list timing of verbalizations, directedness and content of verbalizations and amount of intermediate processing required as factors that affect the consistency and completeness of verbal reports. Thus, while analyzing protocols it is very essential to account for the influence of the above mentioned factors.

Knowledge extraction techniques derived using protocol analysis include Grover's (1983) three phase methodology, delphi (Jagannathan and Elmaghraby, 1985), Crawford Slip Method (Rusk and Krone, 1984) and Smith's and Baker's (1983) idea of presenting experts with novel problems and recording their problem solving process. The common denominator underlying these techniques is an

16-2

#### this.

emphasis towards structuring and systematizing the knowledge elicitation process. Despite it's weaknesses protocol analysis has been used to elicit knowledge in the construction of a majority of the expert systems. Sufficient knowledge maybe elicited using protocol analysis only at the expense of large investments of time, effort and money. This has led many knowledge engineers to conclude that the knowledge extraction phase has significantly stymied the building of expert systems. Grover's method while more structured and streamlined than protocol analysis is subject to all the shortcomings attributed to protocol analysis. This is because protocol analysis is largely resorted to for information elicitation. The development of interactive methods to elicit expertise evidenced a movement away from protocol analysis.

Several methods that elicit expertise interactively have been developed including MDIS (Antonelli, 1983); MORE (Kahn, Nowlan and McDermott, 1985); SALT (Marcus, McDermott and Wang, 1985); PLANET (Shaw, 1984); CAP (Koubek, Salvendy and Dunsmore, 1986) and ETS (Boose, 1986).

#### 2.2 Interactive Methods for Knowledge Elicitation

Expertise Transfer System (ETS) is an interactive grid based method for eliciting knowledge (Boose, 1986). The repertory grid technique was developed for use as a psychotherapeutic tool by George Kelly in 1955 based on his personal constructs theory. Application of this grid approach to diverse domains for information elicitation and several quantitative methods for the analysis of grids are outlined in Shaw (1980, 1981). In fact, PLANET (Shaw, 1984) is an integrated set of computer programs, that offers several alternative approaches for the elicitation and analysis of repertory grids, from one or more people. It provides programs for the analysis of a single grid as well as for comparison and analysis of data across several grids. Boose first applied this technique to knowledge extraction for building expert systems.

ETS is a particularly effective knowledge elicitation tool in analysis type domains, due to the many features it provides including generation of entailment graphs, intermediate and conclusion rules; combination of expertise from multiple sources; presentation of elicited data to the expert in multiple forms; and a system for tracing through the knowledge base of elicited data. NeoETS, a new version of ETS incorporates ideas from Saaty's (1980) Analytic Hierarchy Process (AHP) and uses hierarchies as well as rating grids with multiple variable types to represent knowledge. Hierarchies not only help the expert break down problems into component parts but also permit reasoning at different levels.

MORE (Kahn, Nowlan and McDermott, 1985) is an automated knowledge acquisition system that helps refine an existing knowledge base. The builders of this system based on their experience in hand-crafting MUD, a drilling fluid diagnostic and consultant, developed a qualitative model of causal relations and identified a set of strategies that they found useful in refining (manually) the MUD knowledge base. This causal model and set of strategies are used by MORE to guide the interview process. Given a model of the domain knowledge and the set of strategies this system appears to be particularly effective in refining an existing knowledge base.

SALT (Marcus, McDermott and Wang, 1985) is one of the first systems developed for knowledge acquisition in synthesis type domains where problems are solved by constructing solutions. In particular, SALT was developed as a knowledge acquisition tool for VT, an elevator system configurer. To accomplish this, SALT's problem solving shell draws on three kinds of knowledge (method, constraint and fix schemas) that are elicited and represented using the two subsystems (the interviewing and the rule generation subsystems) within SALT. Based on the input, data driven procedures are used to determine each piece of the configuration and the values for each piece. Knowledge which helps identify constraint violations is drawn upon to spot constraint violations by identifying and checking constrained values. Next, knowledge which indicates "fixes" for constraint violations is used to "fix" potential constraint violations. It appears that the problem solving shell used by SALT facilitates knowledge acquisition in synthesis type domains.

To summarize, the above methods present good but incomplete solutions to the knowledge extraction problem. The PLANET set of programs provide a convenient and structured format to elicit and analyse responses from a single individual as well as compare responses across several individuals. PLANET's suitability as a tool for knowledge extraction may be dramatically improved by incorporating in it a scheme to draw attention to various aspects of the domain and a method to detect conflicts in the elicited data. ETS is very appropriate for eliciting knowledge in analysis type domains; however, in synthesis type domains it's inability to elicit "causal knowledge, procedural knowledge or strategic knowledge", (Boose, 1986) mitigates its usefulness. The methods used in both SALT and MORE may be borrowed to significantly improve existing tools for knowledge extraction. The strategies used in MORE will be an asset in the knowledge refinement stages. The problem solving shell used in SALT will be very effective in eliciting knowledge in synthesis type domains. Below, a conceptual framework that may be used to derive a procedure to elicit knowledge is described.

## 3.0 A Conceptual Framework For Knowledge Elicitation

The conceptual framework outlined here was established by extending a wellknown theory of problem solving and integrating it with a theory which explains how individuals construe themselves and their environment (Figure 1). The former is Newell's and Simon's (1972) theory of problem solving and the latter is Kelly's (1955) theory of personal constructs. In the following section, the two theories are described, and the integration of the two theories, is brought out.

## 3.1 The Problem Space Concept.

In this paper, the authors extend Newell's and Simon's (1972) concept of a problem space. To clarify and illustrate certain concepts, an arbitrary domain is chosen, the control and monitoring of a Flexible Manufacturing System (FMS), by a supervisor. It is asserted that the problem space of any problem consists of an invariant part and a variant part. Further, the authors contend that the invariant part is comprised of the following components: objects, instruments, operator, operations, operating conditions, end products, by products and certain constraints. The variant part is comprised of a set of states. Each of the two parts is described before the relationship between the parts is brought out.

## 3.1.1 The invariant part of the problem space.

Definition of each component of the invariant part. The object is the inputs given to the problem which is being solved. The instruments are the tools, machines

and aids which may be used by the operator to perform operations on the given object. The operator, though a part of the problem space, has an overvlew of it. The operator not only knows what problem is being worked on, but is also aware of what the end products and by products should be, what resources are available or unavailable and under what constraints the work has to proceed. The end products and by products are what would result if the problem is solved successfully. Operations are the actions which may be taken by the operator and which may bring about changes in the other invariant and variant components of the problem space. The invariant components do not change per se, but the values of their characteristics undergo change. This is discussed later, in this paper. Constraints are imposed on the problem space and the operations which the operator may use not only due to the characteristics of the invariant components of the problem space but also due to indirect sources. In a manufacturing situation, factors such as inventory level, batch size and the availability of manpower are some of the direct constraints that affect the operations that an operator may perform. For instance, they may influence the operator's decision to replace a tool. Indirect constraints are imposed on the problem space due to the thought and perceptual process of the operator. How and why this is a constraint is clarified when bringing out the relationship between the variant and the invariant parts of the problem space.

Characteristics and values. Given a problem, each of the components of the invariant parts of the problem space may be defined by a set of characteristics. These characteristics reflect the nature of the component, influence the problem solution process used and reflect how well the problem is solved. These characteristics may take on one of a range of values. The values may be numeric or alphanumeric and may fall in the normal expected range or may lie beyond the limits of this range; thereby, indicating some malfunction in the state of the system. Several or pairs of these values are closely related to each other, i.e., changes in the value of one characteristics. This is due to the interrelationships between the characteristics. Interrelationships between values may also impose constraints on the sets of values that may or may not appear together. Such constraints may constitute the preconditions that have to be satisfied for the occurrence (or non-occurrence) of sets of values.

## 3.1.2 The variant part of the problem space.

Definition and its relation to the invariant part. The variant part of the problem space consists of a set of states. This set is comprised of an *initial state*, a set of intermediate states and a set of final states. For every final state there exists a set of unique intermediate states. It is always possible to define an initial state. However, only a subset of the intermediate and final states can be defined at the beginning although all states can be defined once they are attained. A state is defined by the values and the interrelationships between the values which are taken up by the characteristics of the components of the invariant part of the problem space. A state n changes to state n+1 when some action or operation is performed on state n by the operator. Ideally, there should exist an initial state, a set of defined intermediate and final states and a set of actions. However, in practice this is not true due to the infinitely many combinations in which the values of the characteristics may appear and due to the presence of the human operator in the problem space. This is because the action performed on each state n to reach state n+1 is determined by the operator's perception (patterned or otherwise) of state n. In other words, the action taken and hence the new

state reached depends upon the operator's definition, perception and interpretation of state n.

The second kind of constraint comes into play now. This constraint is due to the perceptual style of the operator. By perceptual style is implied the operator's perception of the characteristics and values (in patterns and schemas versus single discrete characteristics) and the meanings attached to the values. Thus the definition of a state by an operator may be affected by the operator's perceptual style. The set of values and characteristics used by an operator to define a state need not be totally subsumed in the set of actual values which define the state. However, this does not imply that the intersection of two sets is empty. The operator's interpretation of a state refers to the set of outcomes (expected, slightly expected and precluded outcomes), that are predicted by the operator for the given state. In order to reach the most preferred outcome (an outcome at the current step becomes a state at the next step). The performance of an operation merely implies that the values of certain characteristics are changed.

To summarize, it is asserted here that all knowledge required in solving a problem (not the process of problem solution), may be derived by eliciting all characteristics and values associated with each part of the problem space, i.e., objects, instruments, end products, by products, operators, operations and constraints; the set of states which the problem may terminate in and how each of these states which the problem may terminate in are reached (intermediate states, actions). In other words, it is contended that problem solving knowledge may be elicited from an individual by determining how the individual construes the given problem space.

## 3.2 The Theory of Personal Constructs.

Kelly's (1955) theory of a personal scientist or theory of personal constructs indicates that each individual seeks to predict and control events by forming theories, testing hypotheses and weighing evidence. Kelly asserted that individuals' perception of the world, other events, individuals and situations was represented in the form of constructs in the individual. A construct is defined as an internal bipolar scaled dimension which brings out the similarity of a set of elements and the difference of this set of elements from other elements. Thus a construct implies both similarity and contrast. According to Kelly, anticipated outcomes depend on the individual's interpretation of past similar events. This generalization which causes anticipation is based on similarities and differences perceived in events that have already taken place and also on when events are perceived as beginning and ending. This dependence on personal experience accounts for why unique responses maybe given by individuals even when the same set of elements is used for each individual. Constructs may also be represented as hierarchies, i.e., with some superordinate constructs subsuming other subordinate constructs.

Personal constructs theory was first applied by Kelly (1955) to elicit the client's perception of individuals who played an important part in his (the clients) life. Since then the theory has been applied to many a setting (Shaw [1980, 1981]), from helping determine the causes for strained employer employee relations to knowledge extraction for building expert systems (Boose, 1986).

# 3.3 Integration of the Theory of Problem Solving and the Theory of Personal Constructs.

From Kelly's theory that people think in terms of constructs, it is inferred that people can perceive similarities and differences. It is asserted that people can perceive similarities and differences only because they associate characteristics at certain values with given states, events and situations (Figure 1). Individuals anticipate a set of intermediate and final outcomes based on the characteristics which they perceive in a given initial situation, event or state. The choice of action or operation to be performed on the given state (call it state 1), by the individual is determined by the element in the set of anticipated outcomes that the individual wants to reach. This outcome (call it outcome 1) becomes a state (state 2) and the individual performs another action or operation to reach outcome 2, which is state 3. The only distinction between state 2 and outcome 2 (which is state 3) is that the values of certain characteristics are different. It follows that the set of anticipated outcomes will be different for the two states. This continues until the desired final state (equivalently the final outcome) is reached.

In essence, it is asserted that an individual's definition, perception and interpretation of a state is determined largely by the characteristics (and the values of the characteristics) which the individual associates with the state. Thus an ability to elicit from the individual the characteristics, values of these characteristics, the interrelationships between the values, expectations of outcomes and choice of actions for each of the components of the given problem space, should yield detailed problem solving knowledge. This amounts to eliciting how an individual construes his problem space. Different individuals will construe the same problem space differently based on personal experience. Reasons to believe that this framework facilitates the elicitation of detailed knowledge and the derivation of a knowledge elicitation methodology are discussed below.

## 4.0 Implications Of The Above Conceptual Framework

Several direct and indirect benefits accrue from the conceptual framework presented in section 3.0. This framework facilitates the derivation of a knowledge elicitation methodology that possesses several of the attributes listed in section 1.1. This is because the framework provides guidelines regarding what knowledge to elicit and the sequence and format that should be used to elicit this knowledge (Figure 1). Knowledge that should be elicited includes characteristics of components of the problem space, ranges of values of these characteristics, intercorrelations between the characteristics, sets of characteristics that may or may not occur together (and the constraining factors), states of the system, the set of predicted outcomes and the actions that allow realization of a desired outcome, given a particular state.

The repertory grid technique developed by Kelly prompts the elicitation of knowledge in a structured format. Elicitation of knowledge in such a structured format reduces significantly the time required to analyse the elicited data. Further, it permits the use of statistical methods (for example, distance-based clustering techniques) for analysing the elicited data. Boose (1986) and Shaw (1980, 1981) have successfully demonstrated that the repertory grid technique and existing clustering algorithms maybe used to elicit and analyse knowledge, respectively. The components of the invariant part of the problem space may enable the knowledge engineer to draw the expert's attention to various components of problem solving knowledge that the expert may be using (consciously or subconsciously) to solve problems in the domain. This facilitates the elicitation of knowledge that is used subconsciously. Use of Kelly's personal constructs theory makes possible the elicitation of any unique approaches and heuristics that the expert may be employing to solve problems. This is possible because the emphasis is on eliciting how the expert construes the problem space. Although it remains to be demonstrated, there is reason to believe that this conceptual framework may be used to elicit knowledge in both analysis and synthesis type domains; the latter because the methodology emphasizes elicitation of sets of characteristics that may or may not occur together. Implicit here is the elicitation of factors that constrain the simultaneous occurrence of certain sets of characteristics and values.

Indirect benefits accruing from this framework include the scope for gaining insight into the generic components of knowledge within and across domains. Analysis of data using quantitative and qualitative methods and classification of the elicited knowledge based on the components of the invariant part of the problem space may yield valuable insight into the relations, patterns and structures implicit in the elicited data. This may be used to chunk knowledge and establish links between and across the chunks of knowledge in network and frame-based representations.

## References

Antonelli, D. (1983). The application of artificial intelligence to a maintenance and diagnostic information system (MDIS). In the *Proceedings of the Joint* Services Workshop on Artificial Intelligence in Maintenance, Boulder, Colorado.

Boose, J., (1986). Expertise transfer for expert system design. New York: Elsevier.

Davis, R. (1979). Interactive transfer of expertise: Acquisition of new inference rules. Artificial Intelligence, 12, 121-158.

Ericcson, K.A., and Simon, H.A., (1984). Protocol Analysis: Verbal reports as data. MA: The MIT Press.

Gaines, B.R., and Shaw, M.L.G., (1981). New directions in the analysis and interactive elicitation of personal construct systems. In M.L.G. Shaw (Ed.), *Recent advances in personal construct technology* (pp. 147-180). New York: Academic Press.

Graesser, A.C., and Clark, L.F. (1985). Structures and procedures of implicit knowledge. NJ: Ablex.

Grover, M.D., (1983). A pragmatic knowledge acquisition methodology. Proceedings of the Eighth International Joint Conference of Artificial Intelligence (pp. 436-438). Karlsruhe, West Germany.

Jagannathan, V., and Elmaghraby, A. S. (1985). *MEDKAT: Multiple expert* delphi-based knowledge acquisition tool, Technical Report. Louisville: University of Louisville, Engineering, Mathematics and Computer Science Department.

Kahn, G., Nowlan, S., and McDermott, J. (1985). Strategies for knowledge acquisition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-7* (3), 511-522.

Kelly, G.A. (1955). The psychology of personal constructs. New York: Norton.

Koubek, R.J., Salvendy, G., and Dunsmore, H., (1986). CAP: A knowledge extraction methodology for computer programming. In *Proceedings of the Human Factors Thirtieth Annual Meeting.* Dayton, OH: Human Factors Society.

Marcus, S., McDermott, J., and Wang, T., (1985). Knowledge acquisition for constructive systems. In the *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 637-639). Los Angeles: CA.

Moray, N., (1985). Sources of bias and fallibility in humans. In the Workshop on Knowledge Engineering in Industry. Toronto, Canada: University of Toronto.

Newell, A., and Simon, H.A. (1972). Human problem solving. NJ: Prentice-Hall, Inc.

Nisbett, R.E., and Wilson, T.D. (1977). Telling more than we can know: verbal reports on mental processes. *Psychological Review*, 84, (3), 231-259.

Rusk, R. A., and Krone, R. M., (1984). The Crawford slip method (CSM) as a tool for extraction of expert knowledge. In G. Salvendy (Ed.), *Human-computer* interaction (pp. 279-282). New York: Elsevier.

Saaty, T.L. (1980). The analytic hierarchy process. New York: McGraw-Hill.

Shaw, M.L.G., (1980). On becoming a personal scientist. New York: Academic Press.

Shaw, M.L.G., (1981). Recent advances in personal construct technology. New York: Academic Press.

Shaw, M. L. G., (1984). Interactive knowledge elicitation. In the Proceedings of the Canadian Information Processing Society Annual Conference. Calgary, Canada.

Smith, R. G., and Baker, J. D., (1983). The DIPMETER advisor system. In the *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 122-129). Karlsruhe, West Germany.

Swartout, W.R., (1984). Explaining and justifying expert consulting programs. In W.J. Clancey and E.H. Shortliffe (Eds.), *Readings in medical artificial intelligence: The first decade* (pp. 382-398). Reading, MA: Addison-Wesley.

Tversky, A., and Kahneman, D. (1974). Judgment under uncertainty: Heuristics and blases. *Science*, 184, 1124-1131.



Figure 1. Schematic presentation of a concentral framework for builder eligitation to build

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# Design for Acquisition: Principles of Knowledge System Design to Facilitate Knowledge Acquisition

Thomas Gruber and Paul Cohen\* Experimental Knowledge Systems Laboratory Department of Computer and Information Science University of Massachusetts Amherst, Massachusetts 01003

## Abstract

The problem of knowledge acquisition is viewed in terms of the incongruity between the representational formalisms provided by an implementation (e.g., production rules) and the formulation of problem solving knowledge by experts. The goal is to design knowledge system architectures that provide representational primitives that are at the level of the expert's task, rather than the level of implementations. This paper presents principles of *design for acquisition* to guide the design of architectures to facilitate knowledge acquisition. The principles are applied to the design of an architecture for managing uncertainty in medical reasoning, and examples from a working system are shown. The design makes it possible to acquire two kinds of knowledge that are traditionally difficult to acquire from experts: knowledge about evidential combination and knowledge about control. Practical implications for building architectural support software, including knowledge acquisition tools, are discussed.

<sup>\*</sup>This research is funded by National Science Foundation grant IST 8409623 and DARPA/RADC Contract F30602-85-C-0014.

## **1** INTRODUCTION

Knowledge acquisition is the process of gathering knowledge about a domain, usually from an expert, and transforming it to be executed in a computer program. It is a part of the knowledge engineering process, which includes defining a problem, designing an architecture, building a knowledge base, and testing and refining the program. Knowledge acquisition is regarded as the bottleneck in this process. Our thesis is that the design of a knowledge system should anticipate the acquisition process. By analogy with "design for testability," in which digital hardware is designed to be easily tested (Bennetts, 1984), our aim is design for acquisition: designing knowledge systems to facilitate knowledge acquisition.

The first advance on the knowledge acquisition problem was the invention of general architectures: knowledge representation techniques and accompanying interpreters that allow the programmer to encode domain knowledge in a knowledge base separate from the algorithm that interprets it. The EMYCIN architecture is paradigmatic (van Melle, 1979; see also Buchanan and Shortliffe, 1984). Its essential architectural features are a rule formalism with conjunctive premises, certainty factors, and an exhaustive backward-chaining control strategy.

With the general architectures came tools to help the knowledge engineer and expert transform knowledge into the available formalisms. Experts were insulated from the Lisp implementation by rule editors and pseudo-natural language interfaces (Shortliffe, 1976). In TEIRESIAS, Davis (1976) demonstrated that a program can use knowledge about the architecture, such as the structure of rules and the effect of backward chaining, to help users refine and debug the knowledge base.

Recently, the emphasis in knowledge systems research has shifted toward less general, more task-specific architectures (Chandrasekaran, 1986; Clancey, 1985; McDermott, 1983). Many systems share common problem solving methods, despite differences in implementation. When the task can be characterized at a level independent of the implementation, an architecture can be designed to capture the task-specific problem solving knowledge. For example, the HERACLES architecture (Clancey, 1986) is designed to do *heuristic classification*, a common task for knowledge systems.

Knowledge acquisition tools for task-specific architectures can apply knowledge about the kind of problem that the task addresses and the problem solving methods it provides. For example, ETS (Boose, 1984) is a method of acquiring knowledge for hierarchical classification tasks. It applies a psychological theory of how to elicit classification hierarchies from people. SALT (Marcus, McDermott, and Wang, 1985) assists in knowledge acquisition for iterative design tasks such as configuration. The architecture for SALT identifies three kinds of domain knowledge used by its problem solving strategy, and SALT uses knowledge about their form and purpose to focus and constrain the knowledge acquisition dialog. In both cases, the knowledge acquisition method is driven by demands of the task (e.g., classification or configuration) rather than the implementation formalisms (e.g., rules).

Both ETS and SALT acquire knowledge for well-characterized problem solving methods. However, the solutions of many kinds of problems are not well-characterized. For these problems, knowledge systems, and ultimately tools to build them, must be designed. Design means choosing knowledge representations and control strategies that can bring expert knowledge to bear on novel problems. Careful attention to the design of a problem solving architecture can make knowledge acquisition easier both for knowledge engineers and automatic knowledge acquisition tools.

This paper explores principles of design for acquisition, by which the design of a knowledge system can facilitate knowledge acquisition. In Section 2, we present three general principles of design for acquisition. In Section 3, we show how these principles have been applied in the design of an architecture for managing uncertainty in medicine. The architecture makes it possible to acquire two kinds of knowledge that are traditionally difficult to acquire: knowledge about evidential combination and knowledge about control. In Section 4 we show how the principles of design for acquisition imply a hierarchical organization of tools for implementing knowledge system architectures, emphasizing the integration of knowledge acquisition support.

## **2 PRINCIPLES OF DESIGN FOR ACQUISITION**

This section presents three principles that should be considered during the *design* of a knowledge system. They may be familiar to knowledge engineers as heuristics for knowledge representation. We present them in the context of making it easy for *experts* to express their knowledge.

**Principle 1** Design task-level representational primitives to capture important domain concepts defined by the expert.

This principle prescribes that the knowledge engineer provide a language of task-level terms. It addresses a fundamental obstacle to knowledge acquisition, the *representational mismatch* between the way that an expert formulates domain knowledge and the way the knowledge is represented in an implementation (Buchanan, Barstow, Bechtel, Bennett, Clancey, Kulikowski, Mitchell, and Waterman, 1983). For example, knowledge acquisition in a strictly rule-based architecture is ultimately *rule* acquisition; and if it is difficult for an expert to express problem solving expertise as rules, then it is hard to acquire the knowledge. The problem is that rules are implementation-level primitives.

An example of a *task-level* primitive is the notion of a "trigger" – a special relation between data and hypotheses such that when the data are found, a hypothesis is immediately activated. For the cardiologist, a 45 year old man complaining of chest pain with exercise "brings to mind" the hypothesis of angina; trigger is natural construct for diagnosticians. If it is a representational primitive for the system, then acquiring triggering relations from the expert is straightforward. If instead one must achieve the effect of a trigger by, say, "tuning" the certainty factors of rules or the weights on links, then it will be difficult to acquire, explain, and modify this knowledge.

Principle 1 implies that architectures should be designed to support a language of task-level terms for the *expert*, to minimize the effort of translating knowledge into computable form. The

primitives that represent task knowledge should hide the implementation as much as possible; they should provide the illusion of a "virtual machine" whose behavior is described in task-level terms.

**Principle 2** Design declarative representational primitives.

From the standpoint of knowledge acquisition, declarative knowledge representations are preferable to procedural ones. The meaning of declarative representations can be "read" directly, whereas the meaning of procedures can only be had by executing the procedure or simulating its execution. For experts to understand procedural representations of their knowledge they must first understand the algorithm that interprets them. Even when knowledge seems naturally represented with procedures (e.g., control knowledge), formulating it declaratively can facilitate acquisition, explanation, and maintenance.<sup>1</sup> In Section 3.2 we show how designing declarative primitives for control knowledge allows one to represent expertise in deciding what to do next under conditions of uncertainty. By making the knowledge explicit and declarative, the expert can examine the assumptions underlying his control decisions.<sup>2</sup> In Section 4 we show how representing task-level terms declaratively allows the use of conventional "form-filling" user interface technology in knowledge acquisition tools.

**Principle 3** Design representations at the same level of generalization as the expert's knowledge.

This principle can be summarized with two caveats:

- Don't force experts to generalize except when necessary.
- Don't ask experts to specify information not available to them.

Generalization is one of the dimensions of representational mismatch, the distance between the expert's formulation and the implementation.<sup>3</sup> A representation and its referent in the world are at different levels of generalization if there are distinctions in the world that the representation fails to capture or the representation makes artificial distinctions. An example of overgeneralization is forcing a large range of values into a small set of categories. The expert interpreting blood pressure considers the full range of systolic/diastolic ratios, while the knowledge engineer may want to categorize it as high, normal, or low, to make it easier to

<sup>&</sup>lt;sup>1</sup>Neches, Swartout, and Moore (1984) also advocate this principle, emphasizing the advantages for explainability and maintainability.

<sup>&</sup>lt;sup>2</sup>Clancey (1986) also argues for the explicit representation of control knowledge to facilitate explanation, knowledge engineering, and tutoring.

<sup>&</sup>lt;sup>3</sup>Another dimension is *operationalization*, converting advice to procedures, which has been shown to be very difficult to overcome (Mostow, 1983).

implement. Conversely, the knowledge engineer may ask the expert to specify more knowledge than he or she has, again to suit the implementation. For example, the expert may be asked to supply degrees of belief with far more precision that is justified by his or her knowledge. Knowledge acquisition not just building a knowledge base, but getting the system to perform *correctly*; Principle 3 is critical to refining and debugging knowledge systems.

## **3** A CASE STUDY OF DESIGN FOR ACQUISITION

In this section we illustrate the principles of design for acquisition in the context of a medical expert system. We show how the design of the system facilitates acquisition of two kinds of knowledge that are traditionally hard to acquire: knowledge about how to combine evidence and knowledge about how to control the order of actions. The ability to capture this expertise gives the system a unique ability to manage uncertainty by selecting or planning actions that will minimize uncertainty or its effects.

## 3.1 Task domain: Managing Uncertainty in Medicine

MUM is an knowledge system that Manages Uncertainty in Medicine, currently in the domains of chest and abdominal pain. (See Cohen, Day, Delisio, Greenberg, Kjeldsen, Suthers, and Berman, 1986, for details.) Physicians make a distinction between retrospective diagnosis, in which all the evidence is known in advance and the goal is to make the correct diagnosis, and prospective diagnosis, which emphasizes the proper management of the patient through the workup, a diagnostic sequence of questions and tests. In prospective diagnosis, uncertainty about the patient's condition is managed by gathering evidence in the best order (e.g., to maximize diagnostic information and therapeutic effectiveness and to minimize cost and discomfort). MUM's task is prospective diagnosis; it uses expert knowledge about evidence and control to generate an intelligent workup for a patient.

The knowledge acquisition task for MUM includes not only eliciting *heuristic associations* (Clancey, 1985) between evidence and diseases ("What are the symptoms of angina?"), but also *combining knowledge* ("What is the effect of risk factors like smoking on the hypothesis of angina when there has only been one episode of pain?"), and *control knowledge* ("Under what conditions should an angiogram be given?").<sup>4</sup> The expert for MUM has a wealth of combining and control knowledge, central to his expertise as a physician. This knowledge is difficult to represent, and thus acquire, in current architectures. We designed MUM in accordance with the principles discussed above to make it easy to acquire combining and control knowledge.

<sup>&</sup>lt;sup>4</sup>An angiogram is an expensive, invasive test for coronary artery blockage, usually given only after other tests show positive results.

## 3.2 Design for acquisition of combining knowledge

Combining knowledge specifies how belief in several pieces of evidence is combined to support a single conclusion. Remarkably, knowledge engineers rarely ask experts how they combine evidence. Instead, fixed, global numeric functions that compute degrees of belief are built into the architecture (Duda, Hart, and Nilsson, 1976; Shafer, 1976; Shortliffe and Buchanan, 1975; Zadeh, 1975). Although the numeric representations and functions are a convenient implementation formalism, they make it surprisingly difficult for experts to express their knowledge about how they manage uncertainty (Cohen and Gruber, 1985; Szolovits and Pauker, 1978).

Numeric degrees of belief and global combining functions violate Principle 3 because they require experts to *summarize* knowledge about uncertainty, that is, state their knowledge at an inappropriate level of generality. To assess a degree of belief, the expert must consider and summarize several factors, including probability, salience, and cost. Consequently, experts have trouble specifying them. They violate Principle 1 because the numeric degrees of belief do not correspond to domain knowledge (such as frequency statistics) and are combined by functions chosen by the knowledge engineer, not the expert. Uniform combining functions are easy to *implement*, but do not accommodate task-specific methods for combining evidence in different ways.

MUM's design does three things to facilitate the acquisition of combining knowledge. First, it replaces the real-valued numeric representation of uncertainty with symbolic states of belief that are meaningful in domain terms. Second, it provides an explicit representation for clusters of evidence, to encapsulate diagnostically significant subsets of evidence. Third, it replaces the global numeric function with *local combining functions*, specified by the expert, for each cluster of evidence.

MUM represents belief as nominal values that characterize the expert's evaluation of evidential support. Seven states of belief are defined by the expert: confirmed, disconfirmed, supported, detracted, strongly-supported, strongly-detracted, and unknown. They are primitives at the task level; each has diagnostic or therapeutic significance.

MUM represents combinations of evidence with *clusters*, frames that represent diagnostically significant groupings of evidence. With respect to evidential support, diseases are clusters. Clusters also represent intermediate results, such as definitional data abstractions (Clancey, 1985) and common groupings of clinical findings. For example, the cluster chest-pain-when-eating illustrated in Figure 1 describes a situation in which the chief complaint of a patient is pain or pressure in the chest that begins after eating. This cluster *triggers* the disease classic-esophagealspasm. crescendo-pain-long-duration in Figure 1 represents the situation where the pain has been increasing in intensity for more than ten minutes. The cluster discriminates between angina and esophageal spasm: pain from the former usually lasts less than ten minutes.

Principle 1 motivates the design of clusters in MUM. Clusters are easy to acquire because they are knowledge level primitives that represent characteristic diagnostic situations. They have the right "grain size." For example, the expert can state categorically the implications of a subset of findings, instead of relying on the system to calculate a partial match to a complete

# 

Figure 1: Two clusters for diagnosing chest pain

Clusters represent diagnostically significant combinations of evidence. They might play a part in a diagnostic scenario like this: A patient reports an episode of chest pain incited by eating (chest-pain-when-eating); this combination of findings is relevant to many diagnoses. (For instance, it *triggers* classic-esophageal-spasm, shown in Figure 2.) The physician then asks about the duration and time course of the pain. If the report matches the situation characterized here as crescendo-pain-long-duration, the cluster is confirmed. This cluster of symptom descriptions is useful in differential diagnosis; for instance, it *supports* classic-esophageal-spasm and *detracts* classic-angina). In these examples, the combining functions specify necessary and sufficient conditions for clusters to be confirmed; no other belief state (such as *supported*) is relevant.

Name: classic-esophageal-spasm Isa: disease **Triggered-by:** (confirmed chest-pain-when-eating) **Combining-function:** *IF* (or (confirmed barium-swallow=spasm) (confirmed manometrics)) THEN confirmed IF (or (confirmed vasodilator-tx) (confirmed nitroglycerin-tx)) THEN strongly-supported IF (confirmed crescendo-pain-long-duration) THEN supported IF (disconfirmed nitroglycerin-tx)) THEN detracted IF (confirmed chest-pain-short-duration)) THEN detracted IF (disconfirmed barium-swallow=spasm) THEN disconfirmed

Figure 2: Part of a Disease Frame for Classic Esophageal Spasm

The evidential combining function for this disease is made up of rules; each *IF* part specifies conditions on the state of belief in clusters, and each *THEN* part asserts a state of belief for the disease. "tx" means trial therapy; for example, nitroglycerin-tx is confirmed if pain goes away when the patient takes a nitroglycerin tablet. Manometrics and barium-swallow are tests; barium-swallow=spasm is a cluster that is confirmed when the barium-swallow shows a spasm. The triggering function has the same syntax as the left hand sides of rules in the combining function. In this example, when the cluster chest-painwhen-eating (Figure 1) is confirmed, the disease classic-esophageal-spasm is triggered. These combining and triggering functions were elicited by a knowledge engineer working with a physician, in the context of actual cases.

From inspecting the combining function, a planner can infer that the tests (manometrics and bariumswallow) are most diagnostic, since they can confirm and disconfirm the diagnosis of this disease. In the frames representing these tests, however, one will find that they are invasive and therefore costly – to be avoided. Slightly less diagnostic information (strongly-supported, detracted) can be obtained from trial therapies, and even less (supported, detracted) from reports of episodes of pain given by the patient. set of findings. This expertise is difficult to represent in architectures like that of INTERNIST (Pople, 1977), where all the evidence for a disease is grouped together in a single frame, and combined by a uniform scoring technique.

In MUM, evidence combination is completely local; there are no global combining functions. Every cluster has its own combining function. The combining function for the first cluster in Figure 1 is simple: when an episode of chest pain (which may also be described as pressure) is incited by eating then this is a confirmed case of chest-pain-when-eating. No other combination of states of belief in evidence has any affect on belief in that cluster. Diseases, also represented as clusters, typically have more complex combining functions. For example, the frame for classic-esophageal-spasm, with the set of rules that define its combining function, is shown in Figure 2. The combining function maps various combinations of states of belief in data and other clusters to a range of belief states for the disease.

Combining functions are essential ingredients in MUM's knowledge base, and they are acquired from the expert. They are explicit, declarative representations of the evaluation of evidential support that is only implicit in global, numeric combining functions. For example, in MYCIN (Shortliffe and Buchanan, 1975), premises of rules list the evidence that will support the conclusion, but the calculation of how well the evidence is supported is hidden from the expert. Thus there is no natural way to say, "Fire this rule only if the belief in premise  $P_1$  is  $b_1$  and  $P_2$ is  $b_2$ , and then conclude C with belief  $b_3$ ."  $b_1$ ,  $b_2$ , and  $b_3$  bear the same arithmetic relationship to each other in every rule. Applying Principle 3 to the design of MUM allowed the expert to characterize a whole range of situations with different ways of combining their evidence. The precise behavior of each combining function is clear.

Local combining functions are easy to acquire in the context of clusters. Only diagnostically significant combinations of evidence are specified, and only some of the belief states in each constituent piece of evidence are considered. This contrasts with the situation where no local combining function is specified, but every possible combination of belief is possible. We have found that having to specify the combining knowledge explicitly and locally makes knowledge acquisition more efficient, when maintenance and knowledge base refinement are considered. Combinatorial problems are avoided because the space of combinations is very sparse; not every combination of belief in every piece of evidence is relevant in the chest pain domain. This holds advantages for knowledge base refinement and testing. First, every combination of evidence is justified. Second, when test cases are found for which combining knowledge is inadequate, the omission is easily localized to the cluster where the combination function is underspecified. Third, if combining functions produce conflicting belief states for the same cluster, it indicates a situation that the expert had not considered. For example, two rules in a combining function may both be satisfied, where one asserts supported and the other disconfirmed. Since the rules are understood in the context of the cluster, they can be modified or augmented easily to resolve the conflict. This is a major advantage of Principle 3 for knowledge acquisition: when the expert is not forced to generalize combining knowledge beyond familiar cases, errors in the knowledge base tend to be errors of omission.

## **3.3** Design for acquisition of control knowledge

A major part of expertise in prospective diagnosis is the ability to gather data in the proper order, omitting unnecessary tests, asking only those questions that pertain to relevant hypotheses, and prescribing preliminary or exploratory treatment before all of the manifestations of a disease are present. This is *control knowledge* about what to do, rather than what to believe. Traditionally, domain knowledge is acquired without troubling the expert to think about control. Simple control strategies such as forward chaining are implicit in the interpreter, separated from the domain knowledge base, and selected by the knowledge engineer. When these weak methods are inadequate, the knowledge engineer coerces the interpreter to do something more complicated, perhaps by ordering rules or having rules communicate via control flags. Other techniques for specifying control, such as procedural attachment in frame-based systems, are, again, designed and implemented by knowledge engineers largely without consulting experts. But experts have useful domain-specific knowledge about *how* to solve problems.

The problem we faced in MUM was how to represent control knowledge so we could acquire it from the expert. The solution is to ask the expert for the parameters of a domain that affect control decisions, and then ask him to formulate control knowledge in terms of these control parameters. For example, some diseases are more *dangerous* than others; some clinical tests are very *costly*; and some evidence is more *diagnostic*. Control knowledge is easier to acquire in these *task-level* terms, in contrast to *implementation-level* parameters, such as the priorities of tasks on an agenda, or the order of clauses in a rule. Since task-level control parameters are declarative they can be reasoned about by a knowledge-based system, and more to the point, they can be acquired.

Control parameters are a vocabulary for describing situations in which the expert knows what to do. *Control rules* (Davis, 1976), acquired from the expert in terms of control parameters, represent the decision points in diagnosis. Given the evidence that has already been acquired, and the hypotheses it suggests, the diagnostician selects some action, typically to gather evidence for a suspected hypothesis, often by prescribing trial therapy. MUM was designed to represent this decision-making process, so that the expert could specify how it should proceed.

Some control rules specify preferences among alternative actions. For example,

Name: prefer-cheap-to-confirming Conditions: action<sub>1</sub> is potentially-confirming, and action<sub>2</sub> is potentially-supporting, and action<sub>1</sub> costs more than action<sub>2</sub>

Strategy: prefer action<sub>2</sub>

The effect of this rule is to cause the system to favor cheaper actions and sacrifice a little support.<sup>5</sup> Other control rules specify focusing strategy:

<sup>&</sup>lt;sup>5</sup>Features of evidence like *potentially-confirming* are derived from descriptions of the actions and the clusters for which they serve as evidence. An action (e.g., running a test) is *potentially-confirming* if it can result in evidence that contributes to a *confirmed* state of belief in a cluster (e.g., a disease).

Name: focus-on-dangerous-supported-hypos Conditions: hypothesis<sub>1</sub> is supported, and hypothesis<sub>2</sub> is supported, and hypothesis<sub>1</sub> is more dangerous than hypothesis<sub>2</sub> Strategy: focus on hypothesis<sub>1</sub>

This rule directs the attention of the system to the most dangerous hypothesis (e.g., a lifethreatening disease) that has support. That is, the system will search for evidence for and against the more dangerous hypothesis first.

Just as the design of clusters and combining functions give structure to the expert's descriptions of evidential belief, control parameters and control rules organize the expert's strategic knowledge. Control parameters define a space of diagnostic situations, called the control space, distinct from the *belief space* of evidential support for hypotheses. In accordance with Principle 1, both the control space and the belief space are constructed from task-level terms. The representation of the control space is designed to facilitate knowledge acquisition from experts rather than forcing them to abide by implementation decisions that they often do not understand. Adhering to Principle 2, MUM selects actions based on declarative control rules; they describe control decisions in terms of explicit control parameters. While it might seem that control knowledge is economically represented procedurally, as with decision trees (Hannan and Politakis, 1985), such representations compile out the conditions for control decisions (e.g., the costs and potential quality of evidence) leaving a sequence of unexplainable actions. In accordance with Principle 3, the design of MUM does not ask the expert to generalize beyond the diagnostic situations with which he or she is familiar. As combining functions prescribe local combinations of evidence, control rules represent local control decisions. Local control rules have the same relation to global conflict resolution strategies (e.g., "choose the most specific rule") as local combining functions have to their global counterparts (e.g., Bayes' rule). Again the local context facilitates acquisition and makes errors of omission more transparent. When control rules conflict, the cause is missing control knowledge in a particular context. For example, the prefer-cheap-to-confirming rule resolves the conflict between more general preferences rules, one that says "prefer actions that are potentially confirming" and the other that specifies "prefer actions that cost less." The tradeoff is acquired from a particular diagnostic situation.

# 4 IMPLICATIONS FOR THE DESIGN OF ARCHITECTURE SUPPORT TOOLS

In the previous section we emphasized the design of knowledge *representations* to facilitate knowledge acquisition, but the principles in Section 2 also have practical implications for the design of software support for knowledge *engineering*. Specifically, the principles guide the design of *task-specific architectures*. A task-specific architecture integrates particular knowledge representation formalisms and problem solving strategies to perform a well-defined task, such



Figure 3: The structure of a MU knowledge base.

In the MU architecture, objects in a symbolic inference net are connected by inferential relations that propagate symbolic values. For example the potential-evidence relation propagates belief states, such as supported and confirmed. At each node, a local combining function determines the belief state of the current node as a function of the belief states of nodes contributing potential evidence. The control knowledge is used to focus (e.g., decide which clusters to concentrate on) and to choose among possible actions (such as prescribing a test), given the state of the objects in the net and characteristics of actions (tests and treatments are executed by actions).
as hierarchical classification.<sup>6</sup> The point for knowledge acquisition is that task-specific architectures can provide a language of task-level terms to the expert and a way for knowledge engineers to implement these terms declaratively and at the appropriate level of generality, hiding the implementation.

This section presents a hierarchy of knowledge engineering tools for an architecture called MU that is a generalization of MUM. Figure 3 illustrates some of the structure of task-level constructs that MU generalized from MUM; for example, triggering and evidential combination are instances of *inferential relations*, which automatically propagate values through a symbolic inference net.

Tool	Objects in User's View	Software Support
Knowledge Acquisition	Application-specific Terms	(Meta-)Knowledge-based Utilities
Interface	diseases, tests, treatments, questions, intermediate diagnoses, criticality of diseases, costs of tests, efficacy of treatment	language-specific editors and form- filling interfaces, inferential consis- tency analyzer, graphical display for objects and relations
Virtual Machine	Task-level Constructs	Task-specific Reasoning Mechanisms
(shell)	clusters connected by	value propagation functions
	inferential relations combining functions control parameters	pattern matching language
	control rules	rule-hased planner
	preference rankings	decision-making support
AI Toolbox	Implementation Primitives	AI Programming Techniques
(KEE)	rules	rule interpreters
	frames and slots	knowledge base bookkeeping,
		inheritance mechanisms,
		assumption maintenance
	Lisp objects and functions	demon and message passing support
	windows and graphic objects	window system

Figure 4: A hierarchy of knowledge engineering tools to support the MU architecture.

Figure 4 shows the organization of software support for the MU architecture.<sup>7</sup> The three tiers correspond to functional levels. The left column shows the hierarchical relationship among tools. The knowledge acquisition interface is constructed from functionality supplied by the shell, which is built on top of implementation primitives supplied by an AI toolbox. The center column shows the objects that a user would work with at each level; experts would use application-specific terms that are instantiations of task-level constructs, which are in turn

<sup>&</sup>lt;sup>6</sup>Task-specific architectures have been designed for many familiar tasks. Among them are varieties of classification and diagnosis (Bylander and Mittal, 1986; Clancey, 1985; Hannan and Politakis, 1985) and design and configuration (Brown 1985; Howe, Dixon, Cohen, and Simmons, 1986; Marcus, Caplain, McDermott, and Stout, 1986).

<sup>&</sup>lt;sup>7</sup>As of this writing, these tools are under early development.

implemented using primitives provided by the AI programming environment. The right column lists some of the functionality provided by software at each level.

At the base of the hierarchy are the implementation level tools. Instead of Lisp, the primitives are AI programming constructs: frames and slots with inheritance and attached procedures, "worlds" for assumption-maintenance, and graphical displays. The software support is standard technology; we currently use the commercial product KEE.<sup>8</sup> The primary user of these tools is the knowledge engineer. Figure 5 shows an implementation-level view of part the knowledge base for MUM, reimplemented in MU.

The middle level is the shell – the software that implements a "virtual machine" that operates on task-level constructs. Supporting a virtual machine level is a natural application of Principle 1. The shell is a set of tools, some that support runtime operations, such as propagating the effects of data through an inference net, and others that provide an interface for customizing task-level terms (defined by the architecture) for a specific application. Task-level constructs are implemented as objects using the AI toolbox, but can be viewed by the user as primitives.<sup>9</sup> For instance, one can relate data to hypotheses with an evidential relation or a triggering relation without thinking about how those relations are implemented. Figure 6 shows a virtual machine view of part of the evidential support relation for MUM.

The top tier is the knowledge acquisition interface, a set of tools that together present a "user illusion" (Kay, 1984) of a language of application-specific instantiations of constructs provided by the architecture. For example, classic-angina is an instantiation of a cluster, and it is presented to the expert as an object related to other clusters and data by links in a graph of evidential support (such as Figure 6). The primary function of the knowledge acquisition interface is to make it easy for experts to formulate their expertise in the available language. A practical effect of applying Principle 1 is that the language is restricted to task-level terms. This allows one to build a knowledge acquisition tool that can apply specific heuristics for acquiring them, as is done in SALT (Marcus et al., 1985).

When task-level terms are represented declaratively as objects, meta-knowledge (Davis and Buchanan, 1984) about how to acquire task-level terms can be represented as annotations to those objects. This straightforward application of Principle 2 allows one to use simple syntactic techniques to improve the user interface for knowledge acquisition. A surprising amount of leverage can be achieved by using conventional data entry techniques, which we will call *formfilling*, to elicit knowledge from experts.<sup>10</sup> Form filling is a generalization of the "fill in the blank" style of data entry, where each "blank" is labeled and presented in a context. The legal input values are highly constrained and possible values are enumerated when known. On-line help is conveniently accessible, in the form of descriptions of the expected input and examples. For instance, choosing from a menu is a simple kind of form filling (for a single "blank"). A

<sup>&</sup>lt;sup>8</sup>Which is, of course, a trademark and product of IntelliCorp.

<sup>&</sup>lt;sup>9</sup>In our implementation, they are represented as class frames, slots, slot facets, attached demons, and messagehandling methods.

<sup>&</sup>lt;sup>10</sup>Of course this is the "easy" kind of knowledge acquisition. The more difficult problems of designing the architecture and implementing task-level primitives still requires a knowledge engineer.



Figure 5: The Implementation-level View: A fragment of the MUM knowledge base as displayed by KEE.

The objects in the user's view are implementation-level objects: frames, annotated slots, and inheritance relationships. The graph shows a fragment of a hierarchy of frames. They are organized by their implementation. The window on the left shows some of the clusters and diseases (a subclass of clusters) for MUM. The window on the right shows a KEE display of a disease frame. The semantics of slots are defined by the architecture; for example, all clusters have a slot for combining-function, defined in the clusters class frame. prinz-metal is a kind of cluster, and it instantiates its own combining function.



Figure 6: The Architectural View: A fragment of the evidential support relation in MUM.

This lattice shows one kind of inferential relation in MUM, the evidential support relation (potential-evidence). The nodes in the lattice represent assertions that may be believed. The links represent the inference paths that evidence may take; belief in one node is propagated (i.e., from right to left) to other nodes for which it is evidence (that it may support or detract). The expert or knowledge engineer can select nodes to edit them or add new nodes, and the graph displays the evidential context. Similar graphs are available for other inferential relations provided by the MU architecture, such as triggering and treatment efficacy, and each relation may be viewed in both directions. This view of the knowledge base differs from the frame hierarchy of Figure 5 in that the structure represents evidential rather than hierarchical relationships – that is, the structure of the knowledge rather than the implementation.

more sophisticated example is the rule editor shown in Figure 7, a kind of "language-specific editor" for acquiring rules of various kinds in MU. It uses descriptions of task-level objects to constrain the user's input and verify its validity. This technique is similar to the menu-based approach to natural language interfaces (Tenant, Ross, Saenz, Thompson, and Miller, 1983). Form filling is possible because the terms that the expert instantiates are *explicit* (i.e., annotated with meta-level descriptions) and *declarative* (not embedded in procedural expressions).<sup>11</sup> Furthermore, integrating the representations used by knowledge acquisition tools with the shell and the implementation environment is possible because the design of the system anticipated acquisition.

### 5 TOWARDS AUTOMATED KNOWLEDGE ACQUISITION

If the problem of knowledge acquisition is viewed as representational mismatch, the primary contribution of design for acquisition is to make the notation for expressing knowledge more comprehensible and accessible to those with the knowledge. Fortunately, good knowledge representation is also good knowledge engineering. An analysis of successful knowledge acquisition tools (Davis, 1976; Boose, 1984; Marcus et al., 1985; Kahn, Nowlan, and McDermott, 1984; Eschelman and McDermott, 1986) suggests that they satisfy two requirements: to identify the kinds of knowledge to expect from the user, and to provide a functional mapping from user input to implementation primitives. When the underlying architecture supports task-level primitives, the first is accomplished and the second is simplified. Thus design for acquisition facilitates knowledge acquisition by both human and machine.

Yet the problem of knowledge acquisition can go beyond representation and implementation issues. It may be that for some kinds of expertise, it is difficult to design *any* notation comprehensible to the expert that can also be executed. If an expert diagnostician is not accustomed to formalizing his or her expertise, there is no *natural* notation other than the *cases* with which he or she works. For this kind of expertise, induction from examples can be an appropriate acquisition methodology. The learning program transforms knowledge in the form of examples, which alone are inadequate to drive a knowledge system, into more general knowledge of the sort useful to the system.<sup>12</sup>

Control knowledge is an example where a knowledge acquisition methodology can profit from augmenting a good design with induction techniques. Experts who are not familiar with programming may have difficulty writing control rules, even if they are specified in a comprehensible language of control parameters. Experience with MUM has shown that a good way to acquire these rules is by analyzing physicians' workups on actual patients. This has suggested a knowledge acquisition tool that asks about control parameters in the context of decision trees, which

<sup>&</sup>lt;sup>11</sup>This is because the meaning of expressions in the input language can be understood well enough by the interface to prevent invalid expressions. (We believe that) the semantics of procedural languages are more difficult to reason about in this way.

<sup>&</sup>lt;sup>12</sup>If the system only knew about a set of examples, and had no generalizations, it would be the extreme of "brittleness": it would reduce to a lookup table.



Figure 7: The Expert's View: A knowledge-based rule editor instantiated on a combining functions for the evidential relation on a cluster.

The rule editor is a sophisticated instance of conventional data entry technology: form-filling. Each term in the rule editor can be selected with a mouse; they are the "blanks" to fill. In the example, the "forms" are rules comprising a symbolic combination function, computing the belief in the diagnosis classic-esophageal-spasm as a function of several sources of evidence. The syntax of the rules is supplied as a parameter to the editor, and can be seen in the rule template at the top of the window. In this case, the left hand side of a rule is a statement about belief in one of the clusters which serve as potential-evidence for this disease, and the right hand side is always a statement about belief in the diagnosis. The user has selected a belief term from the right hand side of a rule. The metalevel description of combining functions for potential-evidence tells the editor that only members of a class called belief-states are allowed for this term, and so a menu is presented. If the user chooses to create a new belief-state, a form for creating new instances of that class is invoked. are equivalent to graphs of expert workups (Cohen. 1986).<sup>13</sup> Each node in the decision tree corresponds to a decision about what to do next; the tree contains a wealth of *implicit* control knowledge. The role of the acquisition tool is to elicit example decision trees, and to walk the expert through hypothetical cases (paths in the tree), asking for control parameters pertinent to each decision. It could ask questions such as "What factors influenced your decision to do action X instead of Y?" The decision tree is then annotated with these *reasons* for action, and inductive techniques are used to find patterns for generating plausible control rules. The research to implement this idea is currently in progress.

#### 6 CONCLUSIONS

We described three principles of design for acquisition and showed how an architecture was designed so that knowledge about evidential combination and knowledge about control could be acquired from an expert. While conventional expert system architectures make it difficult to acquire these forms of knowledge from experts (or assume that they are best left to the engineer), we find that they are integral to expertise in managing uncertainty in prospective diagnosis. We conclude that emphasizing the *acquisition* of knowledge in the design of an architecture is also good *engineering*; if knowledge acquisition tools are designed with the architecture, they can be integrated with runtime and implementaion-level software. The function of knowledge acquisition interfaces is made easier when the underlying architecture supports task-level terms. Finally, we proposed a technique to address the fundamental limitation of the "intelligent interface" approach to knowledge acquisition. When the expert cannot formulate the necessary knowledge in any notation, then expert-guided induction may facilitate generalization from examples of problem solving. However, the success of induction still depends on whether the knowledge engineer can devise the proper language of generalizations – the right task-level terms.

### Acknowledgments

We are indebted to the researchers on the MUM project: Dan Suthers, Rick Kjeldsen, Mike Greenberg, Jeff Delisio, and David Day, and to Dr. Paul Berman, of the University of Massachusetts Medical School. Thanks for insightful comments from Kevin Ashley, Sharad Saxena, and Larry Lefkowitz.

<sup>&</sup>lt;sup>13</sup>They are also also called diagnostic keys, and are often used to publish diagnostic expertise.

#### REFERENCES

Bennetts, R. G. (1984). Design of Testable Logic Circuits. Reading, MA: Addison-Wesley.

- Boose, J. H. (1984). Personal construct theory and the transfer of human expertise. Proceedings of the National Conference on Artificial Intelligence, Austin, TX, August, 27-33.
- Brown, D. C. (1985). Capturing mechanical design knowledge. Proceedings of the 1985 International Computers in Engineering Conference, ASME, Boston, MA, August.
- Buchanan, B. G., Barstow, D. K., Bechtel, R., Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., & Waterman, D. A. (1983). Constructing an Expert System. In F. Hayes-Roth, D. A. Waterman, & D. B. Lenat (Eds.), Building Expert Systems, Reading, MA: Addison-Wesley, 1983.
- Buchanan, B. G. & Shortliffe, E. H. (Eds.) (1984). Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Reading, MA: Addison-Wesley.
- Bylander, T. & Mittal, S. (1986). CSRL: A language for classificatory problem solving and uncertainty handling. AI Magazine, 7(3), 66-77.
- Chandrasakeran, B. (1986). Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, Fall, 23-30.
- Clancey, W. J. (1985). Heuristic Classification. Artificial Intelligence, 27, 289-350.
- Clancey, W. (1986). From GUIDON to NEOMYCIN and HERACLES in twenty short lessons. AI Magazine, 7(3), 40-60.
- Cohen, P. (1986). Managing Uncertainty. Department of Computer and Information Science, University of Massachusetts. Submitted to the Third Conference on Artificial Intelligence Applications, Orlando, February, 1987.
- Cohen, P. & Gruber, T. (1985). Reasoning about uncertainty: A knowledge representation perspective. Pergamon Infotech State of the Art Report. Also COINS Technical Report 85-24, Department of Computer and Information Science, University of Massachusetts.
- Cohen, P., Day, D., Delisio, J., Greenberg, M., Kjeldsen, R., Suthers, D., & Berman, P. (1986). Management of uncertainty in medicine. To appear in *IEEE Conference on Computers* and Communications, February, 1987.
- Davis, R. (1976). Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases. Doctoral dissertation, Computer Science Department, Stanford University. Reprinted in R. Davis & D. B. Lenat (Eds.), Knowledge-Based Systems in Artificial Intelligence, New York: McGraw-Hill, 1982.
- Davis, R. & Buchanan, B. G. (1984). Meta-level knowledge. In B. G. Buchanan & E. H. Shortliffe (Eds.), Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Reading, MA: Addison-Wesley, 1984.

- Duda, R. O., Hart, P. E., and Nilsson, N. J. (1976). Subjective Bayesian methods for rule-based inference systems. *Proceedings of the 1976 National Computer Conference*.
- Eshelman, L. & McDermott, J. (1986). MOLE: A knowledge acquisition tool that uses its head. *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, August, 950–955.
- Hannan, J. & Politakis, P. (1985). ESSA: An approach to acquiring decision rules for diagnostic expert systems. Proceedings of the Second Conference on Artificial Intelligence Applications, 520-525.
- Hayes-Roth, F. & Lesser, V. (1977). Focus of attention in the Hearsay-II speech understanding system. Proceedings of the Fifth International Joint Conference on Artificial Intelligence.
- Howe, A. E., Dixon, J. R., Cohen, P. R., Simmons, M. K. (1986). DOMINIC: A domainindependent program for mechanical engineering design. International Journal for Artificial Intelligence in Engineering, 1(1), July, 23-29.
- Kahn, G., Nowlan, S. & McDermott, J. (1984). A foundation for knowledge acquisition. Proceedings of the IEEE Workshop on Principles of Knowledge-base Systems, Denver, Colorado, December, 89–98.
- Kahn, G., Nowlan, S. & McDermott, J. (1985). MORE: an intelligent knowledge acquisition tool. Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, CA, August, 581-584.
- Kay, A. (1984). Computer Software. Scientific American, 251(3), September, 52-59.
- Marcus, S., McDermott, J., & Wang, T. (1985). Knowledge acquisition for constructive systems. Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, CA, August, 637-639.
- Marcus, S., Caplain, G., McDermott, J., & Stout, J. C. (1986). Making SALT Generic. Department of Computer Science, Carnegie-Mellon University.
- McDermott, J. (1983). Extracting knowledge from expert systems. Proceedings of the Eighth International Joint Conference on Artificial Intelligence, August, 100-107.
- Mostow, D. J. (1983). Machine transformation of advice into a heuristic search procedure. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), Machine Learning: An Artificial Intelligence Approach. Palo Alto: Tioga, 1983, 243-306.
- Neches, R., Swartout, W. R., & Moore, J. (1984). Explainable (and maintainable) expert systems. Proceedings of the IEEE Workshop on Principles of Knowledge-base Systems, Denver, Colorado, December, 173-184.
- Pople, H. (1977). The formation of composite hypotheses in diagnostic problem solving An exercise in synthetic reasoning. Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 1030-1037.

- Shafer, G. (1976). A Mathematical Theory of Evidence. Princeton, NJ: Princeton University Press.
- Shortliffe, E. H. & Buchanan, B. G. (1975). A model of inexact reasoning in medicine. Mathematical Biosciences, 23, 351-379.
- Szolovits, P. & Pauker, S. G. (1978). Categorical and probabilistic reasoning in medical diagnosis. Artificial Intelligence, 11, 115-144.
- Tenant, H. R., Ross, K. M., Saenz, R. M., Thompson, C. W., & Miller, J. R. (1983). Menubased natural language understanding. Proceedings of the Association for Computational Linguistics, Massachusetts Institute of Technology, June 15-17.
- van Melle, W. (1979). A domain independent production rule system for consultation programs. Proceedings of the Sixth International Joint Conference on Artificial Intelligence, August, 923-925.
- Zadeh, L. A. (1975). Fuzzy logic and approximate reasoning. Synthese, 30, 407-428.

### STRUCTURED ANALYSIS OF KNOWLEDGE

S. A. Hayward - STC Technology Ltd., Six Hills House, London Road, Stevenage, Herts, SG1 1YB, U. K.

# B. J. Wielinga & J. A. Breuker - University of Amsterdam Herengracht 196, 1016 BS Amsterdam, The Netherlands.

# ABSTRACT

Traditional approaches to Expert Systems development have emphasized the use of exploratory programming techniques and moving as quickly as possible from conceptualisation to code. This research demonstrates that implementation independent domain modelling is feasible and useful within the context of a methodology which aims at supporting good software engineering principles as applied to Expert Systems.

# INTRODUCTION

This note is based on research carried out within a collaborative research project part funded by the European Community as part of the *ESPRIT* programme\*. The project is entitled 'A Methodology for the Development of Knowledged Based Systems'. It aims to consider the whole of the lifecycle for development of KBS from the perspective of providng "structured techniques" to support the development process. The work to date has given most attention to the knowledge analysis and acquisition process, but the project is now also moving on to consider specification and documentation for design of system architecture for KBS.

The project constitutes a major European research effort in this field (c80 man years over 5 years) and is the only *ESPRIT* funded work in this area.

The project has so far produced a theoretical basis for the modelling of domain expertise in an implementation independent way and developed tools to support this process. The techniques and tools are currently being applied to a number of experimental system development activities within the project and also for commercial clients.

<sup>\*</sup> The partners are STC plc, Scicon Ltd., SCS GmbH, Cap Sogeti Innovation, the University of Amsterdam and the KBSC (Polytechnic of the South Bank, London). The research on knowledge acquisition and analysis is based on ideas originated at the University of Amsterdam.

# THE RATIONALE FOR THE PROJECT

The traditional approach to KBS development has been essentially experimental. This is in keeping with its roots in the scientific field of Artificial Intelligence research. Thus, for KBS, an application area is chosen because it is hoped that it will shed light on capabilities and techniques of application of AI methods. Development is progressed as far as interest and research budgets permit.

When it then comes to the attempt to apply the technology to commercial systems development there is a clash of objectives. In this case the primary concern is to achieve a system with a prespecified level of capability, and preferably within time and cost budget constraints. In other words we expect to apply the normal criteria for any commercial software development. An experimental methodology cannot expect to satisfy these criteria. This point is fundamental and simply recommending "simpler" applications and labelling the methodology "rapid prototyping" does not overcome it. Doubtless science can be converted to engineering by the gradual accretion of experience, but we would like to achieve the transfer more rapidly and more efficiently. In particular in this case one might hope to learn something from the history of other types of software.

Apart from these pragmatic concerns there are other, more theoretical, reasons for dissatisfaction with a wholly unstructured approach to KBS development. There is a huge conceptual gap between knowledge as expressed by an expert and the encoding of expertise in a software system. This is not in any way to denegrate achievements in knowledge representation theory and languages. These achievements have made the gap bridgeable, but it is still wide. The attempt to transform data from protocols, for example, direct to an implementation language is doomed to misinterpretation or underinterpretation.

In our research we have adopted two related ideas in an attempt to advance our capability. The first is that the separation of software development into phases is applicable to KBS, in particular that one should undertake analysis prior to commitments to implementation detail. This implies that the results of the analysis can be expressed in a way which is independent of decisions concerning system implementation. The second is that knowledge can be analysed at a number of levels and that analysis at an epistemological level is necessary to achieve a full understanding of a domain and to manage the transition from knowlege expressed by an individual (or set of individuals) to knowlege embodied in a software system.

The primary concern of the project in the area of knowledge acquisition and analysis has therefore been to provide techniques and support for such an epistemological analysis. The product of this analysis is then precisely the implementation independent representation required to satisfy our software engineering predilections.

It is perhaps interesting to note that when this project started (about 2 1/2 years ago) this approach was somewhat unusual (Bennett's work on ROGET (Bennett, 1983) was one of the few early pointers for us, together with work in the cognitive modelling tradition). However there are a number of instances where other researchers appear to share similar insights, notably Clancey who has also explicitly championed implementation independent analysis (Clancey, 1983; Clancey, 1985), together with work of perhaps more pragmatic origins (tackling the "knowledge acquisition bottleneck" e.g. Marcus, McDermott & Wang, 1985; Kahn, Nowlan & McDermott, 1985) which has found value in abstracting away from implementation details to representation of "higher level" structures in a domain. More recently we have noted the work of a research group at Tektronix who appear to be developing ideas similar to our own, though from a background in algebraic modelling in the formal software engineering tradition rather than cognitive modelling and knowledge representation (Freiling, Refuss, Alexander, Messick and Shulman, 1986).

# LEVELS OF KNOWLEDGE ANALYSIS

We view the process of knowledge acquisition as one of "interpretation": verbal data (e.g. from interviews with experts or textbooks) is interpreted or mapped onto other representations and structures. Classically in KBS development the attempt has been made to map directly from the verbal data to rules or some such implementation formalism. This is difficult and dangerous (in the sense that misinterpretation or underinterpretation is almost inevitable). We suggest that one can explicitly identify levels of analysis and in so doing provide a basis for supporting the interpretation process.

Sloman, 1980, suggests four degrees of depth at which questions about knowledge can be asked: questions at the *individual* level, at the *conceptual* level, at the level of a particular *formalism* and at the level of *mechanisms* that implement a formalism. These levels closely correspond to the levels of representational primitives for different kinds of knowledge that Brachman, 1979, distinguishes: *implementational, logical, epistemological, conceptual* and *linguistic* knowlege.

For the purpose of mapping verbal data onto knowledge we propose five levels representing a synthesis between Sloman's classification and Brachman's representational levels.

#### Knowledge Identification

This level of analysis corresponds to simply recording what one or more experts report on their knowledge. Although the result may be in a formalised form, the representational primitives on which this formalisation is based are linguistic (in the sense that Brachman uses this term). The same knowledge from different experts may have to be represented differntly, because they use different terminology, or because their knowledge is structured in a different way.

#### Knowledge Conceptualisation

Aims at the formalisation of knowledge in terms of conceptual relations, primitive concepts and conceptual models. The knowledge of different experts, and possibly of different subdomains, is unified within one conceptual framework.

#### Epistemological Analysis

At the epistemological level the analysis uncovers structural properties of the conceptual knowledge, formalised in an epistemological framework. Such a framework is based on epistemological primitives representing types of concepts, types of knowledge sources, structuring relations (such as hierarchical relations, inheritance), and types of strategies.

#### Logical Analysis

The level of analysis applied to the formalism in which the knowledge on higher levels is expressed and which is responsible for inference making.

### Implementational Analysis

At this level of analysis, mechanisms are uncovered on which higher levels are based. The representational primitives are the ones which are normally used when an implementation of an AI programme is described (e.g. matching, testing, slot-filling, etc.)

# THE PRODUCTS OF ANALYSIS

Knowledge analysis undertaken within the framework we propose produces as its major output a four layer model of expertise:

1. Domain Level

Definition of the domain concepts and their static relationships - primarily ISA but others may be relevant, e.g. consists\_of, dependent\_upon etc. This could extend to definition of causal relationships in some domains.

2. Inference Level

Definition of relationships arising in a task context. These are conceived of as dynamic and are expressed in an inference structure. This structure describes only what inferences <u>can</u> be made, not which and how they are made in particular instances of problem solving. The inference structure is defined in terms of meta-classes, which describe the role of domain concepts (e.g. hypothesis, data) and "knowledge sources" i.e. the knowledge elements required to make inferences from concepts in one meta-class to another.

#### 3. Task Level

Specification of how the available inferences can be used to undertake a particular task - defined in terms of the goals of the task. There may of course be many task layers specified on a given inference structure, defining methods to achieve different goals or different methods for the same goal.

#### 4. Strategic Level

Definition of how the task level may be controlled e.g. changing task plans if an impasse is reached. We have conducted relatively little analysis of this level, but in general it would appear to consist of a plan, monitor, repair cycle. We anticipate that even sophisticated Expert Systems will often use a fixed task structure and this layer is defined more for conceptual completeness than practical necessity. It does however help to indicate why certain naive assumptions about Expert Systems may not be realised e.g. graceful degradation at the limits of performance will depend on substantial modelling and implementation of this level, it does not arise for expertise per se.

An example of such a four layer model is given in (Wielinga and Breuker, 1986).

# THE PROCESS OF ANALYSIS

The application of these ideas in practise is not as complex as might be imagined. (This perhaps provides some supportive evidence for the methodology in that a "natural" process may be difficult to describe or characterise but will be easy to use).

Fundamental to our approach is that the knowledge engineer should not have to conduct the analysis in a wholly bottom-up fashion; this is very labourious and probably error-prone. In particular the creation of models as described above is very difficult.; a process of top-down refinement is much less demanding. This, of course, requires the predefinition of generic categories to act as a starting point.

The notion of meta-classes and knowledge sources already incorporates a high level of abstraction. We believe that these can be used to create prototyping inference structures which we term "interpretation models" (because of their role in interpreting the verbal analysis by postulating a certain model on the basis of an initial survey of the domain). The model can then be used as a basis for data gathering, since one has a view of the categories one is looking for. As the analysis proceeds the model may be modified or changed, where the data is found not to match. However, the process is one of top-down refinement and thus provides cognitive support, making it more manageable and controllable.

The analysis process overall is defined as consisting of three phases. In the initial phase, which we have called "orientation" the knowledge engineer becomes acquainted with the domain (and the expert). The major concepts in the domain are identified, together with the scope and complexity of the tasks undertaken. At this stage the knowledge engineer is identifying rather than interpreting data and the analysis is kept to a very general or global level. The main outputs of orientation are a statement of potential system boundaries (there may be more than one) and an estimate of the feasibility of the system(s). A start will also have been made on creating the domain lexicon (listing key concepts) and an initial interpretation model will be selected or constructed. The latter at this point is nothing more than a weak hypothesis. This first phase may be regarded as a preparation for knowledge acquisition proper.

Once the scope of the system has been specified, this defines in broad terms the function(s) of the system and its users. The aim is then to construct a detailed specification of the function the system has to perform. This specification includes the knowledge and strategies employed in the expertise. At this stage, processing of data becomes mainly model-driven. After a definition of the main- and sub-tasks involved in the performance of the system's functions the interpretation model may be refined and moves towards becoming the inference structure for the particular domain/task of interest. This involves fully specifying the meta-classes in terms of domain concepts, and knowledge sources using domain relations; plus defining the task structure adequate for the performance of the desired functionality.

With an interpretation model identified, data can be collected and interpreted from expertise-in-action. Thinking aloud protocols may provide these data more adequately than interviews. Such on-line protocols are preferred to interviews or retrospective data, because there is ample evidence that experts do not necessarily employ the types of strategies and knowledge they may claim to use.

The interpretation model is used in such a way that the data from the domain and expert can be

fitted within the structures provided by the model, thus producing a fully specified description of the expertise in a domain. However, the data may also 'refute' the selected/constructed model, which may motivate revisions of the model. For instance, it may turn out that data obtained from early interviews on the strategies the expert claims to use, are not confirmed by more detailed and valid data collected from analysis of expertise in action. However it may also be noted that one may adopt a more synthetic approach to the definition of the task level, since the available inferences in the domain may be manipulated in a way which does not directly mirror their use by an expert. This may be the primary distinction between cognitive modelling and expert system building. Of course the criteria for assessing the adequacy of the model become harder to define if one takes this approach, but this may be compensated by conceptual simplicity at this level.

These stages are described as problem identification and problem analysis. The basic distinction here is between identification of the various components in the domain, notably the domain concepts, but also such things as the roles and functions of user(s) and expert(s) in the domain; and analysis of the problem solving behaviour in terms of describing these components within a coherent model. These two "phases" are in no way sequential. In practise one observes a highly iterative cycle, but the separation is useful in a normative or didactic sense because it enables the processes of identification (e.g. of behavioural components in a protocol) to be distinguished from the fitting of the identified elements within the framework of a model.

It should be noted that the analysis process must also include an analysis of the environment for the proposed system and of the characteristics of the intended users. These are not emphasized here because we have little to add beyond normal good software engineering practice. However it should be noted that user characteristics will impact on the model of expertise, particularly with regard to the task level, and in that area the user and expert analysis are effectively indistinguishable. Nevertheless there is a component of analysis which should cover the normal concerns of user analysis for defining the required man-machine interface, exception handling, explanation and so on.

The result of all this is full analysis documentation covering models of expertise, user and environment. We believe one should then be ready to proceed to issues of system architecture and design. These issues can now be considered on the basis of a full specification of the domain and the role of the proposed system, and without the need to backtrack over major parts of the knowledge acquisition exercise.

### SUPPORT TOOLS

The techniques described above cannot realistically be carried out without computerised support. We have developed a system for this purpose, known as KADS (for Knowledge Acquisition and Documentation Structuring). This system is built on KL-ONE, which is used not only to record the domain concepts and their relationships but also to record the analysis concepts as defined by the methodology. A large part of a domain analysis is then guided and documented by instantiating these concepts and providing values for their attributes. Other components of the system allow the maintenance of lexicons and glossaries, the production of diagrams, and the analysis of interview transcripts. An illustration of the facilities provided is given by the screen shown in Fig. 1. The system is currently implemented in Quintus Prolog on Sun workstations.

### REFERENCES

- [1] Bennett, J.S. (1983) "ROGET: a knowledge-based consultant for acquiring the conceptual structure of an expert system" Stanford University Memo HPP-83-24.
- [2] Brachman, R. J. (1979) "On the Epistemological Status of Semantic Networks". In N. V. Findler (Ed.) "Associative Networks" New York, Academic Press.
- [3] Clancey, W. J. (1983) "The epistemology of a Rule Based Expert System a framework for explanation", Artificial Intelligence 20.
- [4] Clancey, W. J. (1985) "Heuristic Classification" Artificial Intelligence 27 (3).
- [5] Freiling, M. J. Refuss, S. Alexander, J. H. Messick, S. L. Shulman, S. J. (1986) "The ontological structure of a troubleshooting system for electronic instruments" 1st Intl Conf on Application of AI in Engineering Problems, Southampton.
- [6] Kahn, G. Nowlan, S. & McDermott, J. (1985) "MORE: An Intelligent Knowledge Acquisition Tool", Proceedings of 9th International Joint Conference on Artificial Intelligence.
- [7] Marcus, S. McDermott, J. & Wang, T. (1985) "Knowledge Acquisition for Constructive Systems", Proceedings of 9th International Joint Conference on Artificial Intelligence.
- [8] Sloman, A. (1980) "The computer revolution in philosophy" Harvester.
- [9] Wielinga, B. J. & Breuker ,J. A. (1986) "Models of expertise" Proceedings of European Conference on AI, Brighton.



Figure 1. A typical KADS screen display.

#### Specification of Expertise: . . Knowledge Acquisition for Expert Systems

Paul E. Johnson Imran Zualkernan University of Minnesota

> Sharon Garber 3M Company

The purpose of this paper is to describe a framework for creating a representation of the expertise required to perform a class of tasks. The framework is based upon a procedure employed in research conducted by cognitive scientists and psychologists in order to understand the human problem solving and decision making process. The procedure consists of giving individuals real or simulated tasks to perform and asking them to "think aloud" while they work. The comments such individuals make are referred to as verbal protocols, and the techniques used to analyze these comments are termed, collectively, protocol analysis.

The primary motivation for our paper is to address a fundamental difficulty in knowledge acquisition methods. This difficulty is that the process of human thinking we wish to understand is not available to direct observation. In the case of the human expert, this difficulty is even more severe since the expertise of interest is typically not reportable, due to the

compilation of knowledge which results from extensive practice in a domain of problem-solving activity (Anderson, 1981).

2

One result of the lack of awareness on the part of human experts is the use of what is often called rapid prototyping in the development of an expert system. Through interaction with a system prototype, the expert often reveals expertise not discoverable by other means. While we do not disagree with the basic notion that human experts can reveal knowledge and skill in the performance of a specific task (including prototype debugging), we believe that building a prototype system early in the knowledge acquisition process may carry with it committment to a specific model of thinking (inference process) that does not adequately represent the expertise we are attempting to understand.

We regard a system prototype (e.g., Ramamoorthy, 1984) as a specification of the requirements for a model of expertise. In this paper we shall propose an alternative means of creating such a specification, which is based upon the use of human problem solving data. The virtue of our approach, we believe, is that it permits the construction of a system prototype to be delayed until a more adequate view is obtained of the expertise in question.

In the balance of the paper we shall attempt to do several things. First, we will describe, briefly, the background of protocol analysis methods as these have been employed in the psychological literature on human problem solving, in order to make clear both the strengths and weaknesses of this method of

understanding human thinking. Second, we shall propose a definition of expertise that has evolved from work done at the University of Minnesota over the past several years on the nature of expert thinking. Third, we shall describe a framework for developing a representation of expertise based upon the use of protocol analysis techniques. And finally, we shall present examples of the use of this framework.

# Protocol Analysis

Historically, protocol analysis derives from the attempt by psychologists at the turn of the century to use introspective methods in order to gain an understanding of human mental processes. Such introspective methods typically required intensive training on the part of subjects who then attempted to search their conscious memory for the basis of some task performance. Such efforts fell into disfavor among American psychologists when it was claimed that they distorted the process they were attempting to describe.

In Europe, psychologists used versions of the classic introspective method to augment more standard observational techniques for assessing an individual's thought process. The use of such methods has generally been considered informative by these investigators, particularly in the discovery phase of research where the attempt is made to develop a theory that can later be subjected to more standard verification experiments.

In the 1970's, American psychology moved away from behaviorism and toward mentalism and information processing models of thought. Investigators such as Newell and Simon (1972)

19-2

3

began to use a first cousin of the traditional introspective method, called concurrent protocol analysis, to examine problem-solving behavior. Based upon previous work by deGroot (1965), Newell and Simon asked chess players to think aloud while solving chess game situations. The technique used by Newell and Simon required that subjects think aloud, but not attempt to rationalize their problem solving activities, something these investigators regarded as an essential weakness of the earlier introspective methods.

4

The concurrent protocol technique removed some of the criticism of earlier introspective methods and was responsible for much of the early development of the theory of human information processing. Our objective in the present paper is to show how the techniques developed by early investigators such as Newell and Simon can be extended to the study of human expertise. In the next section we present a definition of expertise that has resulted from our attempts to apply the techniques of protocol analysis to the study of expert thinking in a variety of problem solving domains.

# Definition of Expertise

We define expertise as the operative knowledge that underlies the performance of a task. It is characterized by means of generativity, or the ability to act in new situations, sometimes by the presence of causal relationships among concepts, and usually by the likelihood that a given problem can be solved. Expertise is a property of knowledge, and not a property of the behavior we observe as individuals perform tasks.

The expert and a computational model are two artifacts. Each has a set of capabilities for problem solving in a specific domain. One way to represent the expertise for problem solving is as a set of requirements that have to be satisfied in order to solve problems in a given domain. Consider the following example.

Suppose the objective of an artifact is to solve quadratic equations. One way of characterizing what is required to perform this task is by means of the quadratic formula. The expertise in this case can be represented (at least initially) by the following description. For an equation  $ax^2 + bx + c = 0$ , the quadratic formula ( -b +- Sqrt (b\*\*2 -4ac))/2a is a statement of what needs to be computed to achieve a solution.

The quadratic formula is a rule that generates a certain behavior, namely, solving quadratic equations. The formula displays a certain "competency", i.e., it can solve quadratic equations for integer, real and complex numbers. To be able to use this formula an artifact requires certain abilities. For example, the artifact should know how to divide, multiply, subtract etc. If the artifact is to solve quadratic equations involving complex answers, it should also know how to do complex arithmetic. The abilities implied by the quadratic formula are realized by what Simon (1980) has termed the inner environment of the artifact doing the solving. Addition, for instance, can be implemented in a number of ways, depending upon the internal processing architecture of the artifact performing addition.

5

In the above example, the expertise in question can be characterized by the quadratic formula, plus the abilities needed to realize this rule. Such a characterization is at the knowledge level (Newell, 1981). It is also a description of what needs to be computed (Marr, 1982). Hence, it is a part of a Computational Theory for the task of solving quadratic equations.

The above description of expertise is incomplete, however. In addition to a statement of rules, and the abilities required to realize these rules, we need information about goals and how they are related to the use of abilities. In our example it would be useful to know, for instance, that the discriminant (b\*\*2 - 4ac) is a good determiner of which goals to follow, i.e., (1) if the descriminant < 0, invoke the abilities for complex arithmetic; (2) if the descriminant >= 0, invoke the abilities for real arithmetic; and (3) if the descriminant = 0, the ability to perform square root is not required.

The above statements are examples of rules that define goals for achieving a problem solution. The application of these rules is invariant with respect to the resources used to implement the abilities; that is to say, it does not matter how I do real arithmetic, rule 2 will still hold true. Also, the rules are specific to the "grammar" of a domain, i.e., the quadratic equation (We do not suggest that such rules are, literally, a grammar of expertise, but rather that the idea of a grammar is a useful analogy for thinking about certain consequences of the representation we have created). The rules are essentially a representation at the knowledge level of what needs to be

19-5

6

computed in order for the task of solving quadratic equations to be completed.

Expertise, then, is a representation at the knowledge level which is competent in generating a certain problem solving behavior. A part of this expertise is the set of abilities which are required to realize the solution of problems to which the expertise applies. These abilities are themselves realized in different ways, depending on the artifact performing the task.

Abstractly, expertise is a map from some problem space to a solution space. Hence, an important characterization of expertise is the specification of input and output. We take the specification of input to be a set of legal inputs that a given expertise accepts as valid. Inputs are an important part of expertise because they determine what kinds of features (cues) need to be extracted from the environment for problem solving to be accomplished. The specification of output is a set of legal solutions to problems in a domain. These solutions are the set of "strings" recognized by the "grammar" of expertise. In the next section we give an example of expertise in a specific domain (experimental design) and the methods used to determine it.

We summarize the definition of expertise proposed here as follows. Expertise is defined over some domain of subject matter and class of tasks or problems. It has five parts:

- O Possible solutions and components of solutions to a given class of problems.
- O Specification of "legal" inputs and "legal" outputs for the given class of problems.

7

- 0 Relationships between problem inputs and solution outputs.
- 0 "Legal" ways of moving between solution components, including a specification of the data or cues that must be processed to make such transitions.
- O Abilities required to realize the transitions between components.

We now turn to a description of a framework for obtaining information from the expert, and for developing, from this information, a representation of expertise.

### Framework

We shall assume any problem solving process has two components, a problem solver and a problem. We also assume that the problem solver starts with little or no information and gathers what is needed as problem solving proceeds. We further assume the problem solver has some criterion for determining when a solution is achieved. We also assume the problem solver has a set of abilities that are used in order to solve the problem. This set of abilities defines the "competency" of the problem solver to act effectively in a given domain. For example, in solving algebraic problems, a problem solver might compute the quadratic formula to solve a quadratic equation. Hence, the problem solver has the "competency" to solve a quadratic equation. Finally, we assume that a problem solver proceeds by setting goals and executing actions to satisfy these goals.

Expertise in a domain is a specification of what is required to solve problems in that domain. Protocol is a trace of problem

8

(9-7

solving. If we consider the expert to be a black box, protocol can be considered analogous to the output of a computer program that has the same behavior as the black box. But protocol is more than a trace of a program, it contains explicit statements of what the program is trying to do.

9

Our objective in knowledge acquisition is to construct an artifact (e.g., a computational theory) that solves a class of problems which are currently solved by an expert (or experts). To construct this artifact we need a specification of its requirements. This specification outlines what needs to be computed to solve that class of problems.

There will be a number of artifacts that can achieve the same performance in a variety of ways. The way the expert solves the problem works, in part, because it is adapted to the capacity of the human information processing system (e.g., human algorithms take into account short term memory limitations). Since we may implement our theory on a variety of processors, we need a description that does not depend on a particular processing architecture. The purpose of knowledge acquisition, in this case, is not to learn how to solve a problem, but rather to discover what is required to solve a problem.

A protocol trace provides a record of human problem solving activity. We propose to use this trace to develop a specification of the requirements for any system that attempts to solve the same problem as the system (human expert) giving the protocol. Thus, given a class of problems, and verbal protocol from experts solving these problems, our task is to determine a

method for transforming information in these protocols into a description of expertise.

We now state the problem of knowledge acquisition as follows:

- 0 The expert is an artifact that realizes a certain problem solving behavior by using its expertise.
- 0 The task of knowledge acquisition is to determine this expertise.
- O The expert has developed a set of abilities that are necessary to realize this expertise.

0 We cannot observe directly what the expertise is like.

- 0 We can, however, observe the invocation of the expert's abilities in a problem solving session via a trace of problem solving behavior.
- O Since we can observe the invocation of abilities in the expert, we can have some idea of what the expertise is like.

4

O A statement of the expertise required to perform a task serves as a specification of the requirements for a computational model to perform the task.

We next present a specific methodology for collecting and analyzing protocol data to arrive at our representation of expertise.

# Methodology

A good way of characterizing a problem-solving domain is to look at the problems being solved. In pragmatic terms this involves looking at problems that are given to experts to solve. Sometimes this process can be facilitated by the natural taxonomy

10

(9-9

of problems in the domain. Consideration of the problems being routinely solved then becomes a first set of data points in the problem space to be studied.

A given problem specifies a data point in a problem space. For any such data point we can have a trace of problem solving by an expert. This trace includes two things; a solution for the data point in question, and a trace of what was being done while that solution was achieved.

# Coding of Data

The purpose of coding is to transform data into a form that can be helpful in building a representation of expertise. This would correspond to, for example, taking a set of data points (say ordered pairs) and plotting them on Cartesian coordinates to <sup>--</sup> determine a correlation.

We present, below, a description of components of the coding scheme employed in our recent work on expertise.

0 Actions.

Actions are things that are syntactically indicated by verbs. Actions can include things like collecting data, doing a computation, etc. If the data coder is familiar with the scheme, these can be coded for with reasonable accuracy (We are currently conducting a separate study of expertise in protocol analysis). Coding for actions involves recognizing verbs and coding the sentences associated with these as actions. A coder with some knowledge of the domain will be better at extracting these actions than someone who has no domain knowledge and only the notion of Verbs to go on. We have found that someone who is less

11

than expert in a field often makes a good coder since he/she is familiar with basic vocabulary, but does not tend to extrapolate beyond the available data.

O Episodes.

In the protocol data, there are patterns of actions that are repeated within and across different problems. Such patterns are what we call episodes. These patterns exist together, in time, due to some necessity of performing them in a certain order. The simplest pattern would be a sequence of actions. The assumption is that the pattern of actions in a given protocol is a trace which was generated while the problem solver was attempting to achieve a goal. Different types of episodes correspond to different types of goals.

0 States.

States are problem conditions defined by new data. New data can be further subdivided into environmental data and internal data. Environmental data correspond to the expert seeking some information from the problem domain. Internal data, on the other hand, correspond to data produced via some computation. To code for data, it is necessary to keep track of when new information is articulated in the protocol.

O Goals.

Corresponding to each episode is a goal the problem solver is trying to achieve. We consider each goal as a black box which has a set of inputs and a "functionality" that produces a set of outputs. Knowing the states that are associated with each type of episode, the correspondence between the type of goal and the

12

type of episode, one can usually code for the possible set of outputs. Such outputs are new states generated in that episode. It is considerably harder to code for inputs related to goals since they might be outside the boundary of the context defined by the episode.

0 Abilities.

To achieve a given goal, the problem solver requires certain abilities. For example, to solve a differential equation one generally has to know how to take first and second or higher level derivatives. In general, if A is a goal, then an ability is what is required to perform the actions necessary to achieve that goal.

O Contexts and Decisions.

A problem solver is in a particular context at each stage of the problem-solving process. Contexts are defined by the solution, or part of a solution, that is being considered at each point in problem solving. Decisions are associated with each context. Without loss of generality, we can assume that the objective of a decision in a given context is to include or exclude, in the final solution, parts that correspond to that particular context. Hence, contexts also indirectly specify the decisions that need to be made in a domain. Syntactically, contexts are specified by one or more nouns which are a part of the solution space.

### <u>Analysis</u>

The objective of analysis is to formulate a specification for expertise in a domain of problem solving. Syntactically,

19-12

13

such a specification consists of a set of bubbles. Each bubble has a unique string as its identifier. Each bubble can itself have levels of specification. In addition, there is a set of directed relationships between bubbles represented by arrows. The arrows are of two kinds.

0 Arrows between two bubbles

O Arrows starting from a symbol represented by ==. Each arrow has a "box" associated with it. This box includes a set of strings (see Figure 1). All such strings are members of a finite set called the set of possible cues. Each arrow also has associated with it a "cloud". Each cloud contains a set of strings; all these strings are uniquely determined members of a finite set called the set of possible goals. Finally, each arrow also has associated with it a "triangle". Each triangle includes a set of steps. Each of these steps is a member of a set called the set of abilities.

# Insert\_Figure 1 about here\_

Semantically, the specification of requirements for the expertise in a given problem-solving domain consists of contexts associated with each bubble. The identifier of a bubble is the unique string which identifies the context (the related solution). In addition, there is a set of cues which is the union of all the outputs of all the episodes identified. Finally, there is a set of goals which is the set of all strings identifying goals. Particular assignments of goals to clouds determines the assignment of cues to boxes.

19 - 13

14

Each bubble identifies a context of problem solving. It can be seen as a possible part of the final solution. Relationships between bubbles determine directed "pathways" of problem solving. They determine a way of moving between components of the solution, or of making decisions for a solution. The cloud associated with each relationship identifies the goals that specify the abilities needed to "travel" on that relationship or "pathway". The cues in the box associated with the relationship represent the set of possible "triggers" to activate the abilities (goals). The union of all the boxes is the input specification. The set of all possible subsets of bubbles (at all levels) defines the output.

To create a specification for coded data, we define a bubble for each context. The formation of relationships between bubbles is determined by a sequencing scheme (described below). The assignment of goals is determined by recognizing one or more of the decisions as a primary sequencer of problem solving in a given domain. Said another way, the assignment of goals is determined by "major" decisions in each problem-solving context. Since contexts are related to decisions, these statements are equivalent with respect to the assignment of goals. In determining the primary sequence, a useful notion is that of a major hypothesis that determines the solution. Each such hypothesis defines a problem-solving context. An example in classification tasks (such as diagnosis) would be the set of possible solutions to a set of diseases.

15

Sequencers are goals that are essential to making a decision related to one of the major contexts. For example, a decision cannot be made unless it is considered to be part of a solution. This corresponds to proposing a hypothesis. Similarly, a decision cannot be made until it is evaluated. So "propose" and "evaluate" are two of the major sequencers that determine the inclusion of goals in bubbles. We have used the idea of a context being part of a given solution as a major sequencer. More complicated sequencers can be used, depending on the task.

An essential property of a sequencer is that it divides the protocol coding into temporal intervals. The boundaries of an interval are called dividers. These are determined by a pair consisting of a primary sequencer and a context. Both the sequencer and the divider are said to be associated with the context. The episodes and the associated goals falling in an interval between two dividers determine the relationship between the contexts associated with the dividers.

There are no restrictions imposed on the dividers except via limits of human cognition. This does not corrupt our methodology since two different sets of sequencing schemes/dividers can yield the same assignment of goals to clouds. The intervals can be overlapping. Once the temporal intervals and the boundaries are determined, a relationship is created for each unique interval such that the ends of the interval are the bubbles associated with the contexts in the dividers. All the goals falling between the interval are included in the bubble associated with that relation. This, in turn, uniquely determines the assignment of

16

cues to boxes. If there is more than one goal in an interval, then the content of the box is the union of functional outputs of those goals. Hence, each point in the problem space specifies a set of relationships between bubbles.

We illustrate the above analysis with an example based upon our previous discussion of the quadratic equation. A part of the expertise encoded in the quadratic formula is its "competency" to characterize a solution as either complex or real. We shall attempt to map that part of expertise in the quadratic formula that is responsible for distinguishing between complex and real solutions.

The expertise we are looking for to split the solution space into complex and real spaces is encoded in the square root symbol. The particular use of this symbol, and its semantics in the quadratic formula, define the "competency" for splitting the output space into two spaces. We can further refine the space to reflect the ability to specify equal roots. We represent the expertise as shown in Figure 2.

# Insert Figure 2 about here

We now describe an application of the above methodology to a specific piece of work currently in progress at the University of Minnesota.

### Case Study: Industrial Experimental Design

The problem we wish to consider is often referred to as Off-line Quality Control. It can be described briefly as follows. A client comes to an industrial statistician for help in

17

designing a statistical experiment to study a process. The client is typically an engineer or a scientist-who is interested in improving particular aspects of a system. For an engineer, the problem might be coming up with optimum parameters for a process to create some product. The experiment must be statistically sound and also satisfy certain cost constraints given by the client.

The objective of the statistician's proposed design is to improve the performance of some process. This process can be characterized in terms of an experimental unit that flows through the system over time. The experimental unit is the experimental material to which a treatment is applied in a single trial. The experimental unit may be a typewriter ribbon, a lump of cookie dough, a plot of land, a manufacturing plant or it may be a group of patients in a hospital or class of drugs. The statistician considers certain properties of the experimental unit, such as availability, size, and representativeness of the objects in the experiment.

The statistician is interested in various aspects of the process. These include time and cost of running the experiment, ease in changing factor levels, and blocking information. For example, differences in operators or in amounts of raw materials or in temperature and humidity conditions during the course of the experiment are important blocking considerations in designing an efficient experiment.

At the termination of the process, the measurements made on the experimental unit constitute the values of the response

18
variable or variables. The statistician knows which characteristics to measure and how these are measured.

In the investigation we wish to describe, we examined (using the methodology described above) the expertise of two statisticians solving a series of simulated industrial design problems. Each problem was based upon an actual encounter between an industrial statistician and a client. Our investigation consisted of the following:

#### Selection of Test Problems

The objective here is to determine the extent to which the set of problems we select represents the task domain of interest. In our investigation we were interested in a system capable of performing experimental design in a manufacturing environment. To accomplish our objective, we discussed the problem area with the experts and selecting actual problems in a specific manufacturing domain.

#### Collection of Protocol

Two experts were selected to solve our problems. The first expert is a University Professor, although he has had considerable experience in industry as well. The second expert is from industry (a large manufacturing company). We refer to these experts by initials Ch and KK respectively. The experts were given identical problems and verbal protocols were collected during problem solving. These protocols were analyzed as follows.

19

#### <u>Analysis</u> of Data

We defined a problem-solving episode as a series of goals, together with the actions used to achieve these goals (as described above). Listed below are the major episodes used to analyze the protocol data collected from our two experts performing the experimental design task.

- 0 Understand Product/Process
- 0 Relate Factors to Response
- O Propose Design
- O Probe for Additional Information
- 0 Evaluate Design
- O Consider Experimental Conditions
- O Calculate Sample Size
- O Consider Blocking
- 0 Reduce Factors/Levels of Factors
- O Consider Interactions
- 0 Consider Experimental Error

As mentioned above, it is necessary to determine the way in which the problem solution evolves during problem-solving activity. One means of doing this is to keep track of all the new information that is added to the problem, either by the environment or by internal calculations. We have called these additions of information, problem states, and we assume that each new addition results in a new problem state. Actions add information to the current problem state.

#### Frequency Analysis

Since we are concerned about the "what" rather than the "how" of the problem solving process, one convenient way of representing our data is through simple frequency counts. For instance, we have a record of our two experts solving the same problem and arriving at similar results. The first question we wish to ask, in this instance, is, what are the important goals and abilities required to realize different contexts? To answer this question, we look at each protocol and observe the frequency of occurence of each kind of episode. An example of the data from one such analysis shown in Figure 3 (see list of abbreviations in front of the figures at the end of the paper for a list of the episodes referred to by number in each figure).

#### Insert\_Figure 3 about here\_

Inspection of Figure 3 shows that:

- O Given a particular problem, there is strong correspondence between the kinds of episodes invoked by each expert.
- O Some episodes are consistently invoked more than others across problems. These episodes define a "minimum capacity" for the artifact.
- O For each problem PROPOSE DESIGN is invoked a very small number of times, suggesting that the proposed artifact should have mechanisms for making very few and very good initial guesses.
- 0 A large part of the problem solving process seems to be devoted to EVALUATE DESIGN. This suggests that the artifact

21

should have good evaluation capabilities. In other words, the artifact should be adept at suggesting a design at any point in problem-solving process, and then evaluating it against the constraints posed by the client.

O Although the artifact should have good guessing abilities, the large effort concerning evaluation indicates that the design is done in an evolutionary manner.

A second question that can be addressed by examining the frequency of episodes is to consider the goals that contribute to good first guesses about an appropriate design. To answer this question we can plot the frequency of all episodes before the first guess of the design is made (see Figure 4). From the data in Figure 4, we observe that understanding the product/process, and understanding experimental conditions, seem to play a major role in the selection of a design.

#### Temporal Analysis

A second kind of analysis that can be performed on protocol data is based upon the temporal sequence of occurence of episodes. To do this kind of an analysis we plot in Figure 5 the states of problem solving against episodes (The multiple numbers shown vertically along the abscissa in Figure 5 & 6 result from the program used to produce the chart - the two numbers in each column should be interpreted sequentially, i.e., is 11, is 13, etc). Each point on the graph shown in Figure 5 corresponds to the activation of a particular kind of episode during a problem-solving state. We call this kind of a diagram an episode trace. Figure 5 gives the episode traces for problem 1 for both

22

our experts. Figure 6 gives the episode traces for problem 2 for the same two experts.

### Insert\_Figure 5 & 6 about here\_

Some of the episodes in a problem-solving trace can be used as "anchors" for studying the relationships between other episodes. For example, in Figures 5 & 6, PROPOSE DESIGN is such an episode. We use PROPOSE DESIGN as a primary sequencer. The corresponding intervals, dividers and boundaries for this sequencer are also shown in Figures 5 & 6. Table 1 summarizes the designs proposed by each subject on the two problems plus three additional problems as identified by the primary sequencer PROPOSE DESIGN and the temporal dividers in each problem.

# Insert\_Table\_1\_about\_here

As shown in Table 1, there is correspondence between the kinds of designs proposed by the two experts. We also observe the following repetitive transformations between designs.

> FACT -- Frac FACT SCR -- Frac FACT SCR -- LS

The data in Table 1 suggest a tentative conclusion about the structure that is generating these transitions, namely, Frac FACT, LS and PB are kinds of SCR designs. Also, Frac FACT and LS are kinds of FACT designs.

We now describe how a specification for expertise in experimental design can be developed by using the primary

19-22

23

sequencer and episode traces shown in Figures 5 & 6. For the sake of simplicity, we will show only that part of the solution space consisting of Full Factorial Designs, Fractional Factorial Designs, Screening Designs, Balanced Incomplete Block Designs and Latin Square Designs. We extracted these types of designs by looking for nouns in the protocol that corresponded to parts of the solution. The set of such designs defines the bubbles in our specification; each bubble corresponds to a type of design.

The next step involves looking at the temporal intervals coded in each protocol for each expert, and using these intervals to determine possible transitions between the alternative designs. We illustrate this process by considering problem 1 and looking at the episode trace generated by Ch (Figure 5).

Each temporal interval defines a relationship between the dividers on its boundries. Consider temporal interval #5. Its dividers are (PROPOSE DESIGN, FACTORIAL) and (PROPOSE DESIGN, FRACTIONAL FACTORIAL). We conclude that there exists a relationship between Factorial and Fractional Factorial bubbles in the specification.

We mark an arrow from Factorial to Fractional Factorial in the specification corresponding to this temporal interval. Next, we look at all the episodes that are activated in this interval. These are episode 4 (PROBE) and episode 7 (CALCULATE SAMPLE SIZE). From the protocol we extract the goals related to these episodes. We draw a cloud associated with the relationship between the two designs and include the two goals. Next, we look at the states generated as outputs from each of these goals, and include the

24

states in the box associated with the relationship. Finally, we identify the set of abilities that are required for achieving these goals. We include "the ability to calculate sample size" and the "ability to probe for information" in the triangle of abilities.

25

The above description identifies the process by which a relationship can be determined from a temporal interval. The same process can be repeated for each temporal interval in the two problems. If any of the intervals correspond to a relationship between the same two parts of the solution (and have the same direction), then the union of the corresponding set of 4 goals, set of cues, and set of abilities for each temporal interval, defines the corresponding sets for the relationship that is included in the specification. For example, if in another protocol, a temporal interval with the same dividers as the temporal interval discussed above was observed, and there was one episode, say EVALUATE in the interval, then we would simply add EVALUATE to the cloud. We would also add the states associated with EVALUATE to the box and add the abilities associated with this episode to the triangle associated with the relationship.

We observe from Figures 5 and 6 that the first episode in each interval has a special property; the first divider has an empty context. Such temporal intervals form special kinds of relationships in the specification, namely, the transition is made in the absence of any information. In the specification, the absence of information is represented by an == sign.

The final specification for our example is given in Figure 7. This specification is a "road-map" of the possible transitions between parts of the solution and the design problem. For example, to move from considering a Factorial Design to a Latin Square Design, one needs the ability to "evaluate design" and the ability to "calculate sample size". The map also shows that the discrete nature of variables, number of runs, interactions to be estimated etc., are cues that trigger this transition. We can observe from the specification that the union of all abilities associated with the relations stemming from the empty state is the minimum set of abilities required to propose an initial design. The cues associated with this union are a set of cues that need to be extracted from the environment before a consideration of any part of solution can be made. Finally, the set of all cues corresponds to the vocabulary the expert uses to formulate the problem.

<u>Insert Figure 7 about here</u>

#### Conclusions

In this paper we have presented a framework for describing expertise. When we talk about a specification of requirements for expertise, however, we do not mean that this framework can encode all kinds of expertise. We qualify our idea by saying that our representation can describe expertise in a certain class of tasks, for example, expertise in design or diagnostic tasks (Chandrasekaran, 1983; Clancey, 1985). (As stated earlier, we are also currently using these techniques to determine expertise

19-25

26

in protocol analysis). Since a specification has the "competency" to generate a very large number of possible solutionsequences from a finite set of inputs, we might consider it to be a "grammar" of expertise for a class of tasks. But since our formalism is incomplete, we do not make such a claim.

In concluding, we wish to make two additional points. First, the choice of a sequencing scheme in our framework has implications for the specification of expertise. For instance, a given sequencing scheme determines the contexts and the relationships among the contexts for a class of tasks. Different schemes might lead to different representations of expertise. Our study of expertise in a number of different domains has helped us to identify a set of possible sequencing schemes, but in general the choice of sequencing scheme requires further research.

Our second point is that there is a relationship between the nature of the expertise as practiced by human experts in a class of tasks, and the way that expertise is represented in our framework. For example, human experts often use a small number of cues to distinguish between competing sets of possible solutions (Johnson, et. al., 1981; 1986). The implication of this observation for the representation of expertise is that there should not be a large number of "strings" in the cue box in any relationships between the bubbles corresponding to such competing sets of hypothesis. This implication extends to the expertise for a class of tasks, and for specific domains in that class. The "carry-over" of findings from the empirical study of

27

human experts to our proposed framework also requires further investigation.

Obviously, the work we have described is in progress, and while we are reasonably comfortable with the overall shape of our results, many of the details will undoubtedly change as our investigation continues and our thinking improves. At this point, we claim only that our representation can serve as an initial specification for a computational model of expertise. We hope, however, that it will also provide a guide for further thinking on the nature of one of today's most exciting research frontiers.

28

#### References

. . . . .

Anderson, J.R. et. al. Acquisition of problem solving skill. In John R. Anderson (Ed.) <u>Cognitive Skills and Their Acquisition</u>. Hillsdale, NJ: Lawrence Erlbaum Associates, 1981.

Chandrasekaran, B. Towards a Taxonomy of Problem Solving Types. <u>AI Magazine</u>, Winter/Spring 1983, 9-17.

Clancey William J. Heuristic Classification. <u>Artificial</u> <u>Intelligence</u>, December 1985, Vol. 27, No. 3.

deGroot, A. <u>Thought and Choice in Chess</u>. New York: Basic Books, Inc., 1965.

Johnson, P. et. al. Expertise and Error in Diagnostic Reasoning. Cognitive Science, 1981, 5, 235-283.

Johnson, P., Moen, J. & Thompson, W. Garden Path Errors in Diagnostic Reasoning. In L. Bolc and M.J. Coombs (Eds.) <u>Computer</u> <u>Expert Systems</u>. Springer-Verlag (in press).

Marr, D. Vision. San Francisco, CA: W.H. Freeman & Co., 1982.

Newell, A. The Knowledge Level. AI Magazine, Summer 1981, 1-20.

Newell, A. & Simon, H.A. <u>Human Problem Solving</u>. Englewood Cliffs, CA: Prentice-Hall, 1972.

Ramamoorthy, C.V., Prakash, A., Tsai, W., Usuda, Y. Software Engineering: Problems and Prospectives. <u>IEEE Computer</u>, October 1984.

Simon, H.A. <u>Sciences of the Artifical</u> (2nd Ed.), Cambridge, MA: MIT Press, 1980.

## Table 1

م من الد م

Sequences of Designs Proposed by Two Subjects

in Five Experimental Design Problems

Problem	Subject	Sequence
1	Ch	FACT FACT FACT FACT Frac FACT
	KK	FACT LS LS
2	Ch	FACT BI Frac FACT SCR Frac FACT
	КК	SCR Frac FACT
v		
3	Ch	SO C.M.
	КК	FACT C.M B.B FACT
4	Ch	SCR Frac FACT PB Frac FACT
	KΚ	Frac FACT PB Frac FACT Frac FACT
		LS Frac FACT
5	Ch	C.C Frac FACT
	κк	C.C B.B C.C.

# Figure Captions

Figure	1.	Syntax and Semantics of a Specification for Expertise.
Figure	2.	Specification of a Part of Expertise in Solving Quadratic Equations.
Figure	3.	Frequency of Occurrence of Episodes.
Figure	4.	Frequency of Occurrence of Episodes Before a Design is Proposed.
Figure	5.	Episode Traces for Problem 1.
Figure	6.	Episode Traces for Problem 2.
Figure	7.	Specification of Part of Expertise in Doing Statistical Experimental Design.

# Abbreviations Used in Figures

• •

-

Episode Name	Abbreviation	Number
Understand Product/Process	Unde	1
Relate Factors to Response	Reì	2
Propose Design	Prop	3
Probe for Additional Information	Prob	4
Evaluate Design	Eval	5
Consider Experimental Conditions	Exper	6
Calculate Sample Size	Calc	7
Consider Blocking	Bloc	8
Reduce Factors / Levels of Factors	Redu	9
Consider Interactions	Inter	10
Consider Experimental Error	Err	11

2

•--





F14 -.

(9-33

# protocol 1 problem number 1





4

- ·



Ch protocol 1 problem number 1



KK protocol 1 problem number 1



(9-35

Ch protocol 2 problem number 2



KK protocol 2 problem number 2





fq-37

1.

2

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

#### Knowledge Acquisition for Fault Isolation Expert Systems

Kenneth De Jong Computer Science Department George Mason University Fairfax, VA 22030 and The Navy Center for Applied Research in AI Code 5510 Naval Research Laboratory Washington, D.C. 20375

#### ABSTRACT

For the past few years we have had the opportunity to explore the use of AI and expert system technology in a setting in which diagnostics aids for a large number of complex man-made systems are required. We have built (and discarded) several prototype fault isolation shells in the process of understanding how this technology can be usefully applied. We feel that we have now evolved an architecture which is well suited for this task and incorporates a powerful and highly visual interactive knowledge acquisition system.

This paper will provide an overview of the architecture of our latest prototype, describe our experiences with having Navy labs use the knowledge acquisition system, and summarize our plans to further automate the knowledge acquisition process.

#### INTRODUCTION

The expert system technology from the AI community is being applied to a wide variety of problems including fault isolation in complex man-made systems. The Navy has been interested in and supportive of the development of fault isolation expert systems which can improve the quality of their maintenance and troubleshooting activities. As an example, Navy technicians on aircraft carriers may be responsible for troubleshooting several hundred different (sub)systems for which he/she has had varying amounts of training (frequently little or none). To compensate, the Navy has and continues to invest heavily in automatic test equipment (ATE) to aid or replace these technicians. The quality of the "test programs" which drive these ATE stations varies dramatically in spite of a uniformly high cost to acquire them.

Although it is tempting to leap to the conclusion that one could significantly improve this sort of troubleshooting activity with reasonably straightforward applications of current expert system technology, there are several aspects to the problem which raise significant technical issues. First, with several hundred different systems to maintain, it seems infeasible to think in terms of independently developed expert systems for each one. Rather one thinks in terms of a fault isolation shell providing more general а common knowledge acquisition/representation scheme for use with all subsystems. However, even with this level of generality, there are still several hundred knowledge bases to be built, debugged, and maintained in a context in which there can be considerable overlap and/or similarity in the content of many of the knowledge bases. These observations strongly suggest the development of a sophisticated knowledge acquisition system which can be used to facilitate the construction of a new knowledge base for a specific system in a variety of ways including re-using and/or adapting existing knowledge modules.

Compounding the problem of applying current expert system technology is the fact that, for many of the subsystems being maintained, there is little human expertise in the traditional sense of finding someone who is good at fixing a particular subsystem and capturing his/her knowledge in a set of associative rules. Rather, technicians depend heavily on the structural and functional descriptions contained in the technical manuals of the many subsystems they attempt to maintain. This suggests that simple rule-based architectures are not likely to be sufficient for the task at hand.

For the past few years we have had the opportunity to explore the use of AI and expert system technology in this setting. We have built (and discarded) several prototype fault isolation shells in the process of understanding how this technology can be usefully applied. We feel that we have now evolved an architecture which directly addresses the issues discussed above. In particular, we are using component networks with "local" rule bases as the means of building a "causal" model of the subsystems a technician is required to maintain. This has allowed us, among other things, to develop a highly visual and interactive knowledge acquisition system with access to existing (frequently generic) causal descriptions which can be "pulled in" and integrated into the knowledge base currently under construction. This form of knowledge representation also provides us with a framework for automating the knowledge acquisition process even more in two distinct ways. First, there is a significant amount of "off-line" reasoning that can be done with a causal model to infer (and incorporate) higher level associative rules of the form typically formed by expert technicians when very familiar with particular subsystems. Also, feedback from the fault isolation process can be used to refine, correct, and identify problems in the knowledge base.

The following sections will provide a brief overview of the architecture of our latest prototype (unglamorously named FIS, for Fault Isolation System), describe our experiences with having Navy labs use the knowledge acquisition system, and summarize our plans to further automate the knowledge acquisition process.

#### THE STRUCTURE OF A FIS KNOWLEDGE BASE

A great deal has been written about the design and implementation of diagnostic expert systems in general with much of the initial experience and "common wisdom" coming from applications in the medical domain. Although we benefited greatly from this body of accumulated knowledge, we felt that there were two important properties of our particular task domain that needed to be addressed and incorporated into the underlying design of FIS. First, since we are focusing our activities on fault isolation in **man-made** systems, deeper knowledge in the form of plans, schematics, principles of operation, etc. is available in addition to any attempts at forming a set of high level diagnostic rules. It was clear from the start that good technicians rely heavily on both kinds of knowledge when fault isolating. The second important characteristic of our domain of application was that most of the Navy fault isolation expert systems would have to be built for systems for which there was little human diagnostic expertise or experience, since in the ATE world automatic test equipment is delivered simultaneously with new systems.

These observations led to the design of a multi-level knowledge representation (and associated evidential reasoning mechanism) which is described in more detail elsewhere (De Jong 1984, Pipitone 1984, and Pipitone 1986). For our purposes, a brief description of the structure of the knowledge base will suffice in order to understand the knowledge acquisition issues in this context.





We have chosen to represent deeper knowledge about man-made systems in terms of a qualitative causal network model in which nodes represent replaceable modules, arcs express relationships between modules, and arc labels indicating points at which evidence-gathering tests might be made. Causal knowledge is represented as collections of causal rules attached to modules and test points. Figure 1 gives a partial visualization of a causal model for a simple system containing four replaceable modules, six test points, arrows indicating some dependency relationships (information flow), and several causal rules. Notice that there is no *a priori* commitment to modeling a system at a particular level of detail. The evidential reasoning mechanism works at this level of abstraction regardless of what a replaceable module really is (a sub-system, a card, a gear) and what kind of system is being repaired (mechanical, electrical, optical). In addition, hierarchical relationships are easily represented by treating an object as a replaceable module at one level and as a "system" at a lower level with its own replaceable components.

Qualitative causal rules are attached to this dependency network in two ways. First, each module has a local causal rule base describing how that module behaves in isolation (independent of their placement in a particular system). Figure 1 illustrates one of several such rules attached to module 3. In the electronics domain, module 3 might represent an amplifier and have a collection of causal rules of this sort which are generic and inherited by every instance of a replaceable amplifier module. Figure 1 also illustrates that causal rules can also be attached to test points and generally represent configuration-specific knowledge of the sort an experienced technician might evolve over time as he/she accumulates experience with a particular class of systems. One of the important features of FIS is that it will uses causal rules of this second more global type if to improve the rate at which fault isolation occurs (measured in terms of the number of evidencegathering tests required), but is quite capable of fault isolation in their absence using only rules associated with the local behavior of modules.

There are other kinds of information which, if available, can be included in a knowledge base such as *a priori* failure rates of modules, indications of the relative costs of making tests, and module replacement costs. FIS will use this information if present to improve the rate of fault isolation (measured in terms of costweighted tests). However, FIS does not require such information for its diagnostics activities.

#### THE KNOWLEDGE ACQUISITION PROCESS IN FIS

The preceding sections have provided some insight into the motivation for and structure of a FIS knowledge base. In this section we focus on the activities involved in constructing a knowledge base for a particular system (which in the ATE world is designated as the UUT, the unit under test).

It should be clear by now that, at a minimum, the knowledge engineer must construct a qualitative causal model for the UUT which represents the structure and behavior of the UUT down to the level of "replaceable module" appropriate for the particular task (e.g., board-level maintenance in a communications system). This, in turn, implies that domain experts are not the subject of intensive ruleextraction interviews, but rather are called upon to assist in the construction of causal models. We have found that this role shift for domain experts increases their interest in the knowledge acquisition process and, because the focus is on building a model rather than on the extraction of a frequently ill-defined and unarticulated set of rules, significantly reduces the time required to construct a usable knowledge base.



Figure 2: The Knowledge Acquisition System

Productivity can of course be further enhanced with a knowledge acquisition "front end" to assist in the knowledge base construction process. Figure 2 illustrates the basic knowledge acquisition system components which we provide for FIS. Since these causal models have a strong visual aspect to them, a displayoriented interactive editor is a natural choice for working with a knowledge base. In addition, because large systems are frequently constructed from similar components, provision is made to build up libraries of generic modules which can be "pulled in" and instantiated during the model construction process.

Although it is fairly easy to envision a knowledge acquisition front end which can facilitate network building, assisting in the acquisition of causal rules requires some careful thought, hard choices, and in our case continued experimentation. One very natural point of departure is to exploit the "object oriented" paradigm by building up libraries of generic module hierarchies so that each module instance automatically inherits a collection local causal rules. This can be quite effective

when modeling one or more systems at a level in which there are lots of instances of similar modules. However, one is still faced with providing assistance in building such libraries and one-of-a-kind modules. In this case, we have chosen initially to provide a terse rule language with built-in rule expansion capabilities to minimize the effort involved. However, we are not happy with the fact that this still places a large burden on the knowledge engineer and/or domain expert to provide FIS with reasonably consistent and complete sets of causal rules for modules. We have plans to address these problems in the near future and discuss the strategy briefly in the following section.

At this point in time, acquiring the more global causal rules associated with test points has not been a problem because there have been so few of them provided by domain experts! Even more interesting is the fact that the test point rules encountered are of the type that could have been derived by FIS via compile-time reasoning about the network. This raises some interesting issues and plans discussed in the following section.

#### COMPILE-TIME ACTIVITIES IN FIS

As Figure 2 illustrates, the interactive knowledge acquisition interface in FIS manipulates a high level version of the knowledge base intended to facilitate incremental acquisition. However, as with most other high level languages, there are internal representations which are far more efficient for use during execution (i.e., fault isolation in this case). As a consequence, we have found it useful to build a compilation phase into FIS to effect this transformation. It is also an opportunity to catch a number of rather straight forward knowledge base errors frequently made during knowledge acquisition such as missing or inconsistent causal rules associated with modules. For example, during incremental development of the simple model in Figure 2, one could have indicated that checking the frequency at T6 is a useful evidence-gathering test. Furthermore, the causal rule base associated with module 4 might quite legitimately leave open the possibility that frequency out of spec at T6 may be due to the fact that it is already out of spec at T5. This in turn requires some causal knowledge in module 3's rule base relating to frequency tests. If missing or inconsistent, they are flagged at this point.

Notice that this kind of consistency checking is accomplished by introducing into the compiler some of the evidential reasoning mechanisms used during fault isolation. This raises the interesting issue as to what other benefits might accrue from compile-time reasoning. The analogy we like to use is that of a technician studying the static description of a system and learning something of how it behaves. In a similar sense at compile time, FIS is "seeing" the whole network for the first time and is capable of deriving useful information about the behavior of the network from its static description. There are several interesting directions we are exploring.

The first is the ability to derive automatically from a given causal network, additional high level rules of the sort attached to test points. Such rules typically have the form: "If X is ever known to be true at this test point, then either module Y or Z are faulty". This is the sort of reasoning which, in the absence of such a rule, will have to be re-derived each time during fault isolation. Hence, studying the static structure at compile time can lead to derived rules which can dramatically increase the rate at which fault isolation occurs.

An even more intriguing and exciting use of compile-time reasoning comes from observing how the causal rule bases for unfamiliar modules are derived by domain experts. As one might expect, they generate the necessary rules by studying the sub-structure of a replaceable module which is itself a system describable by a causal network. Generally the lower level modules are simpler, more uniform, and better understood. This in turn suggests the possibility of automatically deriving higher-level module rules from lower level module descriptions. We have just begun exploring this opportunity to enhance the knowledge acquisition process and hope to report on our experiences with it in the near future.

#### DEBUGGING AND REFINING FIS KNOWLEDGE BASES

Even with rigorous compile-time checks there are still possibilities for errors in a causal network which will only show up as incorrect fault isolation behavior. Hence FIS provides the usual sorts of tracing and debugging mechanisms during fault isolation. Here we can and do take advantage of the visual nature of causal networks, displaying them in color and imposing on them graphical indications of how the evidential reasoning mechanism and fault isolation process is proceeding. Combining this with a rudimentary explanation facility in the form of ambiguity sets (which modules are suspect) and causal links (the derived chains of causality which made modules suspects) has produced a reasonably efficient interface for debugging and refining a particular knowledge base.

At the same time it is clear that more rigor testing of the fault isolation behavior will be required for real Navy applications. We hope to further automate this activity in the coming year.

#### CURRENT STATUS AND FUTURE PLANS

FIS as described here was originally written in Franz Lisp on a VAX/780 running Unix. During the past year it has been converted to Common Lisp and ported various Lisp machines and M68000-based workstations. It was demonstrated at AAAI-86 running on a Symbolics 3640.

The Naval Air Engineering Center and Harris Corporation have been working closely with us to evaluate the usefulness of FIS on actual Navy systems. Figure 3 shows one of the typical subsystems of a Doppler radar unit which is serving as an evaluation test bed. In this particular application, technicians and/or ATE stations are expected to fault isolate to the board level of the various sub-systems. Experience to-date has been very favorable and further applications are planned with other Navy Labs as well.



Figure 3: A Subsystem of a Doppler Radar Unit

As indicated in earlier sections, we also have plans to explore ways of improving the current FIS knowledge acquisition capabilities. In addition the improvements in the areas of compile-time rule derivation and automated fault isolation certification, we are planning to explore several possible applications of machine learning. The basic observation is that technicians get better at fault isolation with experience, and so we are looking at ways of fielding FIS with the ability to dynamically make changes to the knowledge base as realistic estimates of *a priori* failure rates of modules and actual fault isolation performance data become available.

#### REFERENCES

De Jong, K., "Applying AI to the Diagnosis of Complex System Failures" (1984), Proc. Oakland AI Conference, pp. 121-122.

Pipitone, F., "An Expert System for Electronics Troubleshooting Based on Function and Connectivity" (1984), Proc. IEEE 1st Conf. on AI Applications, pp. 133-138.

Pipitone, F., "The FIS Electronics Troubleshooting System" (1986), IEEE Computer, July 1986, pp. 68-76. Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# An Intelligent Mixed-Initiative Workbench for Knowledge Acquisition

Gary S. Kahn, Edwin H. Breaux, Robert L. Joseph, and Peter DeKlerk<sup>1</sup>

> Carnegie Group Inc. Pittsburgh, Pa. 15219

<sup>1</sup>The authors acknowledge Al Kepner, and Jeff Pepper who contributed to the development of ideas presented in this paper.

# An Intelligent Mixed-Initiative Workbench for Knowledge Acquisition

#### Abstract

The TEST Knowledge Acquisition Workbench (TKAW) enables knowledge engineers and trained domain experts to interactively build knowledge bases representing trouble-shooting knowledge. TKAW provides both system-directed interrogation and graphic-oriented editors for building up the knowledge base. Novice users of TKAW rely heavily on the interrogation techniques, while experienced users tend to use the direct manipulation of graphic items as a preferred method. TKAW supports knowledge base enhancement by 1) providing the user with the ability to browse quickly for information already in the knowledge base, 2) warning users of suspected information duplication, and 3) aiding the user in the detection and correction of erroneous information.

**1. Resolving the knowledge acquisition bottleneck** There are a number of reasons for the "knowledge acquisition bottleneck". Among them is the need for specialists (knowledge engineers) to intervene between the domain expert who has the knowledge and the system in which it must be embedded. In many organizations there are indeed two types of knowledge engineers - one, skilled in interviewing and eliciting information, the other skilled in the software methodologies required to represent and use the information provided.

Most approaches to automated knowledge acquisition have attempted to economize by removing the knowledge engineering experts. While TKAW's goal is similar, our initial work is aimed at making the knowledge engineer more productive by providing a specialized workbench. However, as workbenches of this kind become more powerful, the distinction between knowledge engineer and domain expert blurs, as domain experts themselves become skilled in using the system building tools provided.

Expert system developers typically find that their knowledge acquisition techniques change with the course of system development. In first approaching a new problem, knowledge acquisition tends to be exploratory. The goal is not only to acquire knowledge but, more importantly, to identify a representational format and control strategy of sufficent power to capture domain-specific problem-solving behavior and domain-specific knowledge.

Once this is done, knowledge acquisition becomes constrained by the target architecture. At this point, the rate at which knowledge can be "pumped into" the system changes dramatically. However, it is still a time-consuming process to interview and code a knowledge base. For this reason, several recent systems, including MORE<sup>1, 2</sup>, and ETS<sup>3</sup>, among others, have looked at ways to interrogate experts under the assumption that a chosen underlying architecture can solve the problems of interest. Learning apprentice systems<sup>4</sup> make similar assumptions. Rather than focus on the difficult problem of characterizing/designing underlying representational and control architecture, such systems simplify the knowledge acquisition problem by mapping domain expertise into an already identified shell.

Certain of these systems have attempted to provide intelligent assistance to users. MORE, for instance, automatically browses the knowledge base for weaknesses and generates suggestions for repairing or strengthening the knowledge base. MOLE<sup>5</sup>, a successor system to MORE, goes somewhat farther in identifying reasons for failures of performance and suggesting ways to correct for problems of this sort. The TEST Knowledge Acquisition Workbench (TKAW), addressed in this paper, continues this line of research.<sup>2</sup> Like MORE, TKAW interviews domain experts, and uses an understanding of how the knowledge base will be used to guide its queries. Like MOLE, TKAW provides tools that aid in knowledge base correction given the results of a diagnostic session. TKAW differs in using more powerful interface technologies to create an environment supportive of mixed initiative system development.

TKAW is designed to enable knowledge engineers and trained domain experts to interactively build knowledge bases representing trouble-shooting knowledge. Within these knowledge bases, it is possible to represent both the causal consequences of component and functional failures (failure modes), as well as diagnostic methods, including the effective ordering of diagnostic tests. The TEST (Trouble-shooting Expert System Tool) inference engine is designed to use knowledge bases of this sort in order to find the cause for malfunctions. TEST is a diagnostic shell developed at Carnegie Group Inc. It is currently being used in the development of several systems.

Before considering TKAW features in more detail, it will be useful to look at the basic features of a TEST knowledge base.

### 2. TEST

TEST uses a semantic network of schematic objects, or frames, to represent its operative concepts.<sup>6</sup> Most critical is the **failure mode**. A failure mode represents a deviation of the unit under test from its standard of correct performance. Failure modes are arranged in a hierarchy. At the top of the hierarchy are observable failures of the entire unit, i.e. behavior that is not to specification. At the bottom of the hierarchy are failure modes of individual components. Failure modes at different levels in the hierarchy can be confirmed or rejected based on particular outcomes of tests. Tests are distinct nodes in the network. A procedure attached to the link between a failure mode and a test tells TEST whether to confirm or disconfirm the failure mode, given a particular test result.

The trouble-shooting task proceeds by focusing on an observed or suspected failure mode. An attempt is made to determine whether the failure mode has occurred in the unit under test. If the failure mode has occurred, or if its status remains unknown, then the possible causes of the failure mode are investigated to see if they have occurred. The search process is guided by an underlying representation of the order in which diagnostic experts explore possible causes for identified failure modes. During this process a trace is generated which makes it easy to provide explanations. Once the diagnosis has isolated the underlying component failures, a repair recommendation is made.

Several other objects may also appear in the knowledge base. These include representations of data gathering activities, decision logic, diagnostic methods, and rules. Rules are used by this system as a means of modifying default diagnostic knowledge. One reason for the need to replace default diagnostic

<sup>&</sup>lt;sup>2</sup>The work described below is very much work-in-progress. Much of the implementation remains to be done. However, many of the ideas have already been explored in prototype form.

knowledge is to handle applications where the population of units being diagnosed do not adhere to a single design but rather encompass a number of identifiable minor design variations. Such rules may after the order in which candidate causes are considered. They may also affect the diagnostic significance associated with each symptomatic failure mode or test result. Rules are also used to detect inconsistencies in input data, and infer the occurrence of failure modes.

The TEST architecture meets three requirements for knowledge acquisition systems:

1) It is rich enough to represent the knowledge required to solved the functionally specified problems;

2) It facilitates mapping from procedural descriptions of problem-solving and declarative characterizations of domain knowledge to the underlying representation by using concepts and relations that correspond to those domain experts use in thinking about their task;

3) It is explicit enough to facilitate browsing and analysis required to provide intelligent guidance to users of the workbench.

TEST is implemented in Knowledge Craft<sup>TM</sup>, Carnegie Group's knowledge engineering toolkit, and Common Lisp. TKAW is implemented using Knowledge Craft interface tools. This allows a tight integration between the workbench and the underlying knowledge representations.

#### 3. TKAW's Approach

TKAW is designed to elicit 7 kinds of information required by a TEST knowledge base:

- 1. failure modes and their causal relations to each other,
- 2. decision logic related to the identification of failure modes;
- 3. data gathering activities to be invoked when there is a lack of certainty about the next failure mode to consider for evaluation;
- 4. test and repair procedures;
- 5. rules which override many different kinds of default assumptions;
- 6. diagnostic methods to use in confirming or disconfirming a failure mode hypothesis;
- 7. a variety of attribute information bearing on the assorted objects in the system;

TKAW provides graphic devices, menus, and system-directed interrogation for purposes of acquiring this information. The design approach taken to TKAW, however, is best understood by considering its functional capabilities in terms of a cube with dimensions representing knowledge engineering activities, types of analysis, and source of control.



#### Figure 3-1: TKAW design

In order to be effective, TKAW must provide features which enable the user to effectively create, extend, and modify a knowledge base. Creation refers to what can be called clean slate activities: that is, the initial description of a knowledge base in a manner that is unconstrained by earlier knowledge engineering activities. Extension covers knowledge base enhancements to a pre-existing system. Here, issues of consistent reference, duplication avoidance, and intelligent facilitation become pronounced as new information must be properly placed within the existing system. Modification refers to non-monotonic changes in an existing knowledge base, that is deletions or revisions.

The *type of analysis* dimension highlights the availability of tools for analyzing the knowledge base (static), and for analyzing a diagnostic session (dynamic). The *source of control* dimension conveys that TKAW's behavior and analytic activities can be controlled by either the user or TKAW itself.

TKAW interfaces are designed for modeless operation, enabling the user to move smoothly between operations, as well as to request system guidance. In addition, TEST programs can be run as a separate task, allowing a smooth transition between knowledge base editing and the running of the diagnostic system.

TKAW provides both system-directed interrogation and graphic-oriented editors for building up the knowledge base. Novice users of TKAW rely heavily on the

21-4

interrogation techniques, while experienced users tend to use direct manipulation of graphic items as the preferred method. These methods support each other. For the novice user, a graphic map of the knowledge base is displayed and incrementally augmented as TKAW's questions are answered. The experienced user can directly manipulate the graphic representation, as well as activate the interrogator by selecting menu options and mouse clicking on relevant portions of the graphics display.

The essential core of the knowledge base is displayed as a tree of failure modes (global view). The user selects, at any given time, which of the failure modes in the tree should be expanded, as a focused view. The focused view shows the tests, repairs, and rules associated with the selected failure mode. Each of these objects can be selected for further expansion. On selection, the object is displayed with its attributes and their values in the schema window. These values may be modified within the window. (See figure 3-2.)

The remainder of this paper examines the use of TKAW in more detail. The following section focuses on TKAW as a mixed initiative workbench, and describes the manner and conditions under which users are likely to rely on TKAW's direction. In subsequent sections, the focus is on TKAW facilities that aid in the extension and modification of existing knowledge bases.

#### 4. TKAW's Mixed-Initiative Environment

To be successful, mixed-initiative systems need to

- 1. offer a full continuum of control options between total system control and total user control,
- 2. facilitate transfer among the various control options, and
- 3. allow the user to vary the degree of control appropriately with improving skill level or changes in the task being performed.

The following walk-through illustrates the use of TKAW in terms of these goals. For the purposes of demonstration, we have chosen a non-proprietary example of a knowledge base. The partial knowledge base which we use is for the (highly unlikely) application of on-site diagnosis of central air conditioning systems. (See figure 4-1.)

#### 4.1. A session with a novice

When a user enters TKAW, the graphic windows show only a single node of the tree that is about to be built. That node represents the buffer itself, so that everything can be graphically "attached" to the buffer. Unless told otherwise, TKAW starts off in system-directed mode, providing verbose commentary.

Prompts start to appear in the prompt window and an initial dialogue begins: TKAW first elicits a list of symptomatic problems, considered as functional failure modes.

Computer: Please list all problems to be diagnosed in this portion of the



Figure 3-2: Screen Layout for TKAW


Figure 4-1: A Partial Knowledge Base for Central Air Conditioning Systems

knowledge base.

For more information on valid reponses, strike the HELP key on the keyboard. For more information on using TKAW in general, use Control-HELP.

Problems to be diagnosed:

- User: Hot-Air No-Air
- Computer: [Graphically displays failure modes Hot-Air and No-Air. Hot-Air is in reverse video, indicating that it is now being selected as the basis for further questioning. A copy of Hot-Air is placed in the focused view, but since Hot-Air has no definition yet, nothing else appears within the focused view.]

Because the knowledge acquisition process must be constrained by the interviewee's knowledge and intended focus, TKAW is flexible about the way it obtains its information.

Computer: Unless you indicate otherwise, we will proceed to complete the knowledge base by first asking for the structure of the knowledge base and basic information. Afterwards, we will go back and obtain more detailed information, such as descriptions of the tests and repair procedures. If you wish to change this mode of operation, use Meta-m.

Please describe in words what the failure mode Hot-Air indicates.

*User:* The vents are producing air, but the air is not cold.

*Computer:* What are the possible causes of Hot-Air?

*User:* Low-Freon Broken-Cooling-Unit

Computer: What are the tests for Hot-Air? If several tests may be used, and each test in and of itself is indicative of Hot-Air, please list them all. If several tests must *all* test positive, then that is one *test procedure* so just enter the name of the test procedure.

User: Hot-Air-Check

*Computer:* If there are any repairs that can be performed upon identifying a Hot-Air failure, please list them now. If there aren't any, just strike the RETURN key.

User: [RETURN key]

In the above interaction, TKAW has focused on eliciting the essential structure of the knowledge base: namely, the failure mode hierarchy, and the names of repairs and tests associated with each failure mode. TKAW proceeds in a depth-first, partially elaborative manner. After Hot-Air is selected as the focus, TKAW asks about its causes. Following this, it attempts to get elaborative information about the tests and repairs which bear on the Hot-Air failure, and are represented in the focused view window. TKAW proceeds to ask about Low-Freon, one of the causes of Hot-Air.

As information is provided, TKAW represents it graphically. The symptomatic failure modes were displayed under the buffer node; their proximal causes were represented in the global window as they were provided. Test and repair information was displayed as attached to the focal failure mode in the focused view window. This failure mode is highlighted in the global view to remind the user of the context in which additional information will be interpreted. Thus, the user can both see the system's interpretation of the information provided and follow the developing context in which further questions are asked. This latter point can significantly increase the user's ability to stay with a line of questioning.

The interrogator seeks to complete each object or concept (represented as a schema) that it knows about by filling in values for all the attributes (slots). To perform its task, the interrogator uses the detailed textual view mentioned above. This view is called the *schema window*, since it lets the user view and edit the contents of schemata. As the user is being prompted for a given slot, the interrogator displays that slot within the schema window, if it is not already there, and then it highlights the slot name. The user is then prompted using the prompt associated with that slot name. While the slot is a terse name for referencing a given attribute, the prompt is sufficiently descriptive for any novice. Thus the novice learns slot names by watching the interrogator highlight each slot name in turn as it asks its questions.

Interrogation is controlled by a script which may be modified by the user to change the order in which the information is collected by TKAW. For example, the user

can request that the information in the knowledge base be collected by expanding the tree breadth first instead of depth first. He may also request the immediate collection of detailed attribute information.

Because, as we shall see below, an experienced user can alter the flow of control, it is possible that questions in the script might be passed without being answered or even asked. The system keeps track of such questions and tries to get back to them whenever it can. Prior to terminating a session, TKAW will give the user the opportunity to fill in information not yet provided.

There are several advanced features in the interrogator, such the editing of the previous value in response to the current question, which are available to experts and novices alike. A list of these features is readily accessible by requesting help.

#### 4.2. A session with an experienced user

Using TKAW over time is a learning experience. By providing reactive graphics, TKAW teaches the user the graphic manipulations that can be performed to accomplish what had initially been done by answering questions as they arose. By suggesting, from time to time, where the user can override system control, TKAW informs users of the flexible functionality that is at their disposal.

More advanced users move easily between graphics manipulation and reliance on the system to provide guidance through prompting. When they are first learning to take control, users tend to suspend TKAW interrogation only temporarily. A user may do so simply by invoking a command instead of answering the pending question. This is useful for such activities as selecting another failure mode as a focus of attention. If the new failure mode doesn't exist yet, the user may create it and then select it. Of course the user may also issue any other command he chooses, including commands for browsing, editing, or changing control parameters. This technique of gaining temporary control is illustrated below:

- Computer: If there are any exception rules, please list the names of the rules here.
- User: [Clicks the mouse on the Failure Mode icon to activate the create failure mode command.]
- Computer: Creating a failure mode...

Name of failure mode to create:

User: Blows-Fuse-Often

Computer: [Creates a dummy failure mode on the screen near where the mouse currently sits and attaches the failure mode to the nearest node in the tree. (As the user moves the mouse, the dummy failure mode moves in the global window and it is continually attached to the node that is closest at the time.)]

Position the failure mode with the mouse and click-left when done.

User: [Positions the dummy failure mode as the rightmost child under the buffer name and then clicks the mouse.]

Computer: [Displays the actual failure mode, labeled Blows-Fuse-Often, at the appropriate location on the screen, redisplaying the screen to keep the tree "balanced". Since the command is now complete, the interrogator once again resumes control:]

What are the possible causes of No-Air?

- User: [Clicks on the Blows-Fuse-Often failure mode, once again interrupting the interrogator. This activates the select node command, with Blows-Fuse-Often as its argument.]
- Computer: [Acts on the user command, then returns control to the interrogator. Since the Blows-Fuse-Often failure mode has been selected, and since neither the interrogator nor the user has filled in any information for that failure mode, the interrogator continues by asking the first question in the script for the failure modes:]

What are the possible causes of Blows-Fuse-Often?

Note that the interrogator's behavior has effectively been altered by the actions of the user. Questions that would have been asked according to the script have, at least temporarily, been bypassed.

After a while, many users will find system-directed interactions to be annoying, even though the user can always override the system's requests. Such users can turn the system control off. When the user does so, the computer will immediately inform him as to how to turn control back on. Requesting system help at any time will also tell the user how to turn the system control back on. When the user has answered all questions for the **Blows-Fuse-Often** schema, he is once again given control, until he requests another **fill schema** command or reverts back to systemdirected mode.

#### 4.3. A Synopsis of the Mixed-Initiative Environment

To build a knowledge base with TKAW, the user can do it himself using the graphics editor and command options, or he can let the system drive the task through directed prompts. System interrogation, however, can always be escaped by executing simple commands. Commands may be invoked by typing the textual command, by bringing up a menu and selecting the command from that menu, or by striking a short-hand control-key sequence. Many commands let the user directly manipulate the graphic representation of the knowledge base. Other commands permit viewing of the knowledge base or changing the flow of control. For example, in the middle of being questioned about one portion of the knowledge base and select a different node in the tree, thereby requesting that investigation be diverted to that portion of the knowledge base.

## 5. Extending the knowledge base

Extension covers knowledge base enhancements to a pre-existing system. Here, issues of consistent reference, duplication avoidance, and intelligent facilitation become pronounced as new information must be properly placed within the existing system. TKAW helps users locate and cross-reference objects already in the knowledge base, as well avoid the duplicate creation of objects already in the knowledge base.

TKAW meets these needs by 1) providing the user with the ability to browse quickly for information already in the knowledge base 2) warning users of suspected information duplication and 3) aiding the user in detection and correction of erroneous information. In what follows, TKAW features are described, and then illustrated in terms of a user session aimed at the expansion of the air conditioner knowledge base.

#### 5.1. Features

TKAW features exploited during the extension of the knowledge base can be characterized in terms of the type of analysis on which they depend (i.e. static vs. dynamic), and the source of control (i.e. user vs. system). The illustrated crosssection of the cube displays 9 features explained below the figure 5-1.



Figure 5-1: TKAW support for extension

## Features using static analysis

## User initiated features:

#### 1) Multiple browsing options:

Browsing facilities are required when the user wants to confirm that a new (to be added) failure mode is not already in the knowledge base or to find a similar failure mode which can be copied and edited to create the new object. In a large knowledge base, it is impractical to browse without a means of directing attention to relevant segments of data.

TKAW provides for string search, network browsing, and schematic pattern matching. String search is used to match against the names of knowledge base objects. Search of this kind is constrained by allowing the user to specify the type of object (Failure mode, test, etc.). Network browsing is used to examine objects within the knowledge base that are in a specified relation to a specified object. Pattern matching is used to find objects with attribute values specified in the search template.

#### Both System and User Initiated:

#### 2) Redundancy detection

When developing a knowledge base, the user may unknowingly duplicate a failure mode by using a different name to refer to one already known. TKAW monitors failure modes which have identical causes and consequents, as well as those that share the same test or repair procedure. The user can request this information on demand or ask TKAW to provide an immediate warning when potential redundancy is found.

3) Recognizing failure mode classes

Domain experts typically cluster groups of failure modes in classes. Diagnostically, it is typically sound to determine the type of cause prior to determining its specific nature. Similarly, it often useful to point the diagnostic search toward a class of problems. Class references can also refer to subnets within the knowledge base; this can reduce the amount of information that needs to be reentered.

TKAW monitors for groups of failure modes that typically co-occur as potential causes of other failure modes. The user is provided the opportunity of designating these as classes. TKAW also monitors for references to failure modes in known classes. The user can request this information on demand or ask TKAW to provide an immediate warning when potential class membership is found.

#### 4) Error detection

TKAW monitors for certain kinds of errors. Of particular interest are objects that are not properly linked into the knowledge base. In addition, failure modes without causal, test, or repair information are noticed. These errors are flagged by TKAW upon termination of a session. The user can also request this information on demand.

## Features using dynamic analysis

## **User Initiated**

#### 5) Editing Descriptive Text

During diagnostic problem-solving, system errors may occur in the text displayed to the user. This might include descriptions of failure modes, procedural information in regard to test or repairs, or prompts used to query the user.

Rather than force return to the editor, TKAW allows the user to select text strings for editing in a pop up window. The knowledge base is updated appropriately, as TKAW maintains an association between displayed strings and their knowledge base source.

#### 6) Determining current knowledge

When a diagnostic session is paused, the system may be asked to reveal what it knows or has already concluded. This allows the expert to notice when the system is not making use of relevant information. If the information is lacking or is improperly linked in the knowledge base, this can be corrected. If the information should have been inferred, an inference rule may be added.

#### 7) Determining goals

Similarly, when the running system is paused, it may be asked to reveal both candidate failure mode hypotheses, as well as planned tests or user queries associated with the confirmation or disconfirmation of these failure modes. Thus, users have the opportunity to notice when the system has failed to conclude inferrable information, or is preparing to determine needless information. In addition, hypotheses which should have been rejected on the basis of available information, or not even considered as reasonable candidates can be noticed.

#### 8) Setting Breakpoints

Breakpoints may be associated with failure modes, such that the system pauses automatically when a particular failure mode becomes a focus of attention. At this the point the system can be stepped with explanation; or paused for a return to the workbench. This relieves the user of carefully monitoring a diagnostic run until a point of debugging interest is reached.

#### System Initiated

#### 9) Finding Erroneous Branches

When the system makes an erroneous conclusion, and is subsequently informed of the correct conclusion (already in the knowledge base), the system responds by indicating where the diagnostic search failed to take a branch that would have led. to a correct conclusion. The user may then modify this node.

#### 5.2. An Example

The following example demonstrates how some of the facilities described above can be used to extend the air conditioning knowledge base. In particular, the example illustrates the use of multiple browsing techniques, breakpointing, determining current knowledge and goals, and in-place correction of text.

Prior to this session, the knowledge base has the failure mode tree pictured in 4-1. At this point, Sam, a new knowledge engineer is assigned the task of adding in additional failure modes. He wants to add the failure mode, **compressor-motor**, but is not sure if it is already in the knowledge base. He does a string search on motor, broken, and cooling across failure mode names, but does not find the failure mode that he wants. In order to assure himself that the failure mode is not there, he uses string search to look for a test with compressor in its name. Finding **compressor-circuit-test**, he uses the network browser to look at all the failure mode linked to it by the **test-for** relation. This turns up the failure mode failure mode failure mode is not to be what Sam had in mind. Sam then links it to the broken-cooling-unit. Multiple browsing techniques provided for successful identification of a failure mode already in the knowledge base, thus preventing unintended duplication.

In order to determine the effect of this addition, Sam sets a breakpoint on **compressor-engine**. Sam runs the knowledge base to see how it functions. When this failure mode becomes the focus of attention, the system pauses. At this point, Sam wants to know what information the system knows. The system pops up a window indicating that it is trying to determine if there has been a compressor engine failure, and that it has determined that the air being produced is hot, that freon levels are ok, and that the cooling unit is not broken.

From this, Sam realizes that the system does not know if the thermostat works. He notes that a failure mode is missing so he goes back and adds that failure mode to the knowledge base, by placing an icon in the global window and answering prompt information provided by the system. The knowledge base now appears as in figure 5-2.

Sam then continues to run the example. The compressor enginer is determined not to work. The repair procedure is then displayed in the window but there is a mistake. Sam selects the repair text and modifies it in the middle of a run to correct the mistake (see figure 5-3). This correction is then saved in the data base.

#### 6. Modifying A Knowledge Base

Modification refers to deletions and revisions in an existing knowledge base, that is non-monotonic changes. Here, TKAW provides help by pointing users to parts of the knowledge base that may be impacted by their modifications.

TKAW monitors for several kinds of changes. Some of these include:

1) Reordering of failure mode list

Failure modes which should be considered as the causes of another failure mode (F) are listed in F's "due-to" slot. The order of the list indicates the order in which these failure modes ought to be considered. When modifications are made in this list, the system needs to determine if the reordering is meant to be universal, or is only intended in certain cases. In order to determine this, TKAW (on user request) reminds the user that F is considered as the cause of a set of problems (all the





failure modes in which F appears in their due-to list). The user is asked to indicate if the reordering is independent of the diagnostic context; or is contingent on other information that may be tied to the occurrence of particular causal consequences of F.

#### 2) Renaming failure mode

When a user renames a failure mode (F), there is an ambiguous state. Should the operation be treated as a selected copy and edit, or as a local slot modification? In other words, is the failure mode to be treated as a new cause of the consequent failure mode displayed over the selected instance of F, or should the name change propagate across all references to F. On user request, TKAW provides a list of all the objects in which a reference to F occurs, and permits the user to select where references to F should be renamed.

3) Renaming tests and repairs

TKAW similarly monitors for changes in the names of repairs and tests. Whenever name changes are made, users can request the name change to propagate universally to all references, or get a list of objects in which a reference to the modified entity is found. The user can then indicate where references should be changed.

## 7. Conclusion

By providing a mixed-initiative multi-task environment, TKAW allows knowledge engineers and domain experts the option of directing or being directed by the underlying knowledge acquisition system. By providing a graphic interpretation of the underlying knowledge base as it changes in respect to user input, users are provided with a context in which to understand the impact of replies to system prompts. TKAW supports knowledge base enhancement by 1) providing the user Conclusion: Compressor Engine is broken

The repair is: Install a new compressor engine. To install the engine connect the engine's black wire to the power source and the red wire to ground.

Install a new compressor engine. To install the engine connect the engine's red wire to the power source and the black wire to ground.

engine's black wire to the power source and the red wire to ground.

## Figure 5-3: Correction Of Information During A Test Run

with the ability to browse quickly for information already in the knowledge base, 2) warning users of suspected information duplication, and 3) aiding the user in detection and correction of erroneous information.

## References

- 1. Kahn, G.S., Nowlan, S., McDermott, J., "Strategies for knowledge acquisition", *IEEE Pattern Recognition and Machine Intelligence*, September 1985.
- 2. Kahn, G.S., Nowlan, S., McDermott, J., "MORE: An intelligent knowledge acquisition tool", *Proceedings of International Joint Conference on Artificial Intelligence*, 1985.
- 3. Boose, J., "Personal construct theory and the transfer of human expertise", *Proceedings of the National Conference on Artificial Intelligence*, Austin, Texas, 1984.
- 4. Michell, T.M., Mahadevan, S., Steinberg, L.I., "Leap: A Learning Apprentice System for VLSI design", *Proceedings of International Joint Conference on Artificial Intelligence*, 1985.
- 5. Eshalman, L., McDermott, J., "Mole: A knowledge acquisition tool that uses its head", 1986, Draft.
- 6. Carnegie Group Inc., *Knowledge Craft Technical Manual*, 1986.

## Heuristics for Expertise Transfer: An Implementation of a Dialog Manager for Knowledge Acquisition

Catherine M. Kitto and John H. Boose Knowledge Systems Laboratory, Boeing Advanced Technology Center Boeing Computer Services, P.O. Box 24346, Seattle, Wa., 98124

#### ABSTRACT

One of the most difficult and timeconsuming activities in constructing an expert system is the process of knowledge acquisition. Our objective is to identify a set of heuristics for *expertise* transfer based on our experience in knowledge acquisition for expert systems and to formalize this knowledge as rules. Aquinas, a knowledge acquisition workbench, contains tools to interview experts, analyze, test, and refine knowledge, and generate knowledge bases for expert system shells. A set of heuristics for knowledge acquisition has been defined and incorporated in the Dialog Manager subsystem of Aquinas to provide guidance in the knowledge acquisition process to domain experts and knowledge engineers.

The implementation of the Dialog Manager is described and an example transcript shows the interaction of Aquinas, the Dialog Manager and the expert. A preliminary classification for knowledge acquisition heuristics is proposed. Finally, issues in formalizing strategies for knowledge acquisition and a plan for future research is presented.

#### INTRODUCTION

Aquinas is a knowledge acquisition workbench consisting of an integrated set of tools for automated knowledge acquisition, representation, and reasoning (Boose and Bradshaw, 1986). One of these tools, the Expertise Transfer System (ETS) interactively interviews an expert, analyzes the information entered, assists the expert in refining that knowledge, and generates a production rule knowledge base. Vocabulary, conclusions, problemsolving traits, trait structures, trait weights, and inconsistencies are elicited. This problem-solving knowledge is stored in a rating grid (shown later in Figure 4). Problem solutions (elements) are requested and displayed across the top of a grid, while solution traits (constructs) are listed down the side of the grid as bipolar scales.

The interviewing methodology used to construct the rating grid is derived from Personal Construct Psychology (Kelly, 1955; Gaines and Shaw, 1981; Shaw and Gaines, 1986). The expert supplies traits to discriminate between groups of problem solutions (elements). Each element is given a rating by the expert showing where it falls on the trait scale. Production rules are generated from the ratings in the grid and can be tested in end-user consultations. Knowledge bases can be created for a number of expert system shells (KEE, S.1, M.1, OPS5, etc.) from the common internal representation in Aquinas (Boose, 1985, 1986).

More than 500 knowledge-based system prototypes have been generated by ETS, during the three years of its use at the Boeing Company. A typical prototype

can be constructed in less than two hours.

Aquinas extends the problem-solving and knowledge representation capabilities of ETS by allowing experts to structure information in hierarchies using multiple variable types. Several methods exist for representing and reasoning about uncertainty, and dozens of tools related to building and maintaining the knowledge base are available. A novice Aquinas user is led through an initial interview, but can then be overwhelmed by the large number of alternative knowledge development and refinement methods.

To assist novice and intermediate users in effectively using Aquinas, a **Dialog Manager** was created (Kitto and Boose, 1986). The Dialog Manager is a subsystem of Aquinas that contains heuristics for the transfer of expertise and for the use of Aquinas during the knowledge acquisition process.

#### APPROACH

Our goal was to identify heuristics used by experts and knowledge engineers in constructing a knowledge base using Aquinas. We were concerned with how and why users decide to invoke Aquinas commands in specific situations. We identified these heuristics and embedded them in the Aquinas Dialog Manager.

Another objective was to identify differences between strategies used by the novice user of Aquinas and the experienced user who has acquired skill in using the tool. Could the successful strategies of the experienced Aquinas user be captured and used to guide a novice through a knowledge acquisition session? Could these heuristics be presented to the novice in an explanatory or tutoring capability? Would strategies that prove successful for a particular expert in a specific problem-solving domain be equally successful with a different expert and different domain?

As a starting point, a knowledge engineering development effort using ETS was examined in depth. A team of knowledge engineers had developed a "Financial Advisor" knowledge-based consultant to select appropriate investments for a client. The team maintained a journal summarizing discussions with the domain expert and their experience in using the ETS knowledge acquisition tool.

This detailed account of applied knowledge acquisition was a productive source of heuristics for knowledge acquisition and strategies for the use of ETS. It was supplemented with transcripts and interviews with the domain expert and knowledge engineering team. We reconstructed the interaction between ETS and the domain expert from the stored transcripts. The version of ETS in use at this time did not contain the Dialog Manager, so all the documented heuristics dealing with the use of ETS and the knowledge acquisition process stemmed directly from the knowledge engineers and domain expert.

Next, we repeated the entry of the "Financial Advisor" knowledge base, this time using Aquinas. We were able to compare the performance of the Dialog Manager in selecting Aquinas operations with the original transcripts using ETS without the Dialog Manager. We identified differences in knowledge acquisition strategies and areas where the Dialog Manager's heuristics could be improved.

#### IMPLEMENTATION OF THE DIALOG MANAGER

The Dialog Manager has been implemented as an expert system. Its domain knowledge is the effective use of Aquinas. A rule-based approach to implementation was selected because of its flexibility, maintainability, and modifiability. Heuristics are encoded as rules directly within the Aquinas system.

#### **Options for Dialog Manager Assistance**

The Dialog Manager offers three modes of interactive assistance: *automatic*, *assist*, or *off*.

In the "automatic" mode, the Dialog Manager applies knowledge acquisition heuristics to determine the "best". Aquinas operation to be performed next within the context of the current state of and the the knowledge base characteristics and preferences of the expert. The Dialog Manager displays the recommended command sequence, and an explanation of what it is doing and why it has chosen that option. This mode is most commonly used for new Aquinas users, particularly in the initial interview when they may be unfamiliar with Aquinas commands and desire guidance. The expert may interrupt the Dialog Manager to regain control.

The Dialog Manager also provides limited guidance to the expert in an "assist" mode in which the Dialog Manager again applies heuristics to suggest several suitable command alternatives and display its recommendation. However, experts may overrule the Dialog Manager by replacing the recommendation with their own command choices. If the user overrides a proposed suggestion, the Dialog Manager asks for a textual explanation to be stored in a file accessible to Aquinas developers for later analysis. The recommendation of the Dialog Manager and the command preferred by the user are noted in the permanent history and flagged for the Dialog Manager *learning facility*.

Finally, experts may prefer to use Aquinas with the Dialog Manager in "off" mode. The Dialog Manager makes no suggestions to the expert although it continues to record a history of selected operations. The "off" mode is typically used for brief demonstrations, or by experienced Aquinas users who require little guidance.

Users can also select the *level of prompting* and explanatory information provided by the Dialog Manager. Three levels are provided: *verbose, concise* and *terse,* permitting users to select detailed or abbreviated explanations and justifications for Dialog Manager recommendations.

Dialog Manager Modes		
Control	off assist automatic	No recommendations; Record keeping only. Offers recommendations; User may override. Automatically implements recommendations.
Explanation Level	verbose concise terse	Paragraphs of explanation Medium level of explanation (sentences). Little or no explanation of recommendations.

Table 1. Dialog Manager offers a range of assistance and explanation.

# Data Structures for Representation of Knowledge Acquisition Heuristics

It was necessary to develop special data structures to capture information used in the knowledge acquisition heuristics. Since many heuristics for knowledge acquisition involve temporal reasoning or user characteristics, three types of data structures were defined to record and represent this information:

• the "recent" events list, a mechanism for retaining Aquinas command sequences with associated time stamps. This is stored in memory as an Interlisp record. As a new command is entered, it is added to the recent events list, and the least recent is removed.

• the permanent history, a file containing the complete chronology of Aquinas commands entered by the expert for the current knowledge base. This is stored as an Interlisp file which is loaded into Aquinas when the expert enters Aquinas.

• the user profile, a file containing user model information on individual Aquinas domain expert users and classes of Aquinas users.

#### **EXAMPLE TYPESCRIPT**

The following example illustrates how the Dialog Manager employs its heuristics for knowledge acquisition to guide an expert in developing the "Database Management System (DBMS) Advisor" and provides explanations of its strategies. Recommendations from the Dialog Manager are highlighted in boldface type and preceded by [DM], while Aquinas user entry is in underlined bold type.

In Figure 1, the expert has specified that the Dialog Manager run in "automatic" mode, supplying "concise" explanations of alternatives considered and suggested actions. In this mode, the Dialog --- AQUINAS SYSTEM ----

Would you like to work on an OLD knowledge base, or create a NEW one? AQ\*\*<u>NEW</u>

Would you like help using AQUINAS from the DIALOG MANAGER? AQ\*\*<u>YES</u>

Would you like the DIALOG MANAGER to make suggestions (ASSIST) or run automatically (AUTOMATIC)? AQ\*\*<u>AUTOMATIC</u>

When giving advice, should the DIALOG MANAGER be VERBOSE, CONCISE, or TERSE? AQ\*\*<u>CONCISE</u>

Please give a short description of the problem you are trying to solve. AQ\*\*DBMS SELECTION

Figure 1. Aquinas interviews an expert to obtain problem-solving knowledge. The Dialog Manager will apply heuristics about the knowledge acquisition process and the use of Aquinas to assist the user in constructing a knowledge base. The user has asked the Dialog Manager to automatically guide the interaction.

Manager will control the interaction between expert and Aquinas, prompting the expert when domain knowledge is needed. The user may regain control of the dialog by escaping to the executive after any Aquinas prompt.

Next, in *figure 2*, Aquinas asks the expert for an initial set of DBMS problem solutions. The solutions are then combined into triads (sets of three), and the expert is asked to identify discriminating traits. The Dialog Manager is automatically guiding the user in this interaction. Prior to each step, the Dialog Manager displays a concise explanation of the activity to be performed.



Figure 2. AQUINAS asks the expert for an initial set of potential solutions. The solutions are combined into triads (sets of three), and the expert enters discriminating traits. The Dialog Manager directs the dialog automatically. Its comments are flagged with "[DM]."

The Dialog Manager continues to guide the user, prompting the user in *figure 3* to rate each DBMS solution on each trait scale. In this case, the Dialog Manager suggests several appropriate alternatives, chooses "fill in ratings", and provides a brief explanation of why that option was chosen. *Figure 4* illustrates the Aquinas screen showing this initial DBMS knowledge base. The top window depicts the developing case, expert, solution, and trait hierarchies in a *hierarchy map*. The middle window shows the DBMS *rating grid*, and the bottom window displays implications between traits in an *implications graph*. These are used later to generate rules in the knowledge base. The Dialog Manager automatically displays these graphs after the expert has entered solutions, trait pairs, and ratings for the Database Management System Advisor.



Figure 4. Aquinas screen showing the developing hierarchies, the rating grid, and an implication analysis for the DBMS knowledge base after the expert has entered solution elements and attribute trait pairs. Ratings appear in the rating grid as they are entered. In Automatic mode, the Dialog Manager determines when to update the displays of these graphs.

In our example, the Dialog Manager has been operating in "automatic" mode with "concise" explanations. At this point, however, the Dialog Manager applies the following heuristic:

if fill-in-rating-grid-complete and all-displays-current and current-command-state = exec and dialog-manager-mode = automatic and user-AQUINAS-experience = high

and no-of-current-command-choices >7

then recommendation = OPTIONS DIALOG.MANAGER ASSIST

because "you are an experienced AQUINAS user and many knowledge base refinement commands are appropriate here"

This is an example of a Dialog Manager heuristic based upon user experience and current state of knowledge base development.

The expert has now entered the initial DBMS rating grid and the Dialog Manager is operating in "assist" mode (Figure 5).

Once an initial set of elements, traits, and trait ratings is entered as in our DBMS example above, the expert may choose from more than 150 alternative commands on menus and submenus in Aquinas, including the following activities: to apply analytic techniques to the knowledge base; to edit cases, experts, elements, traits, ratings; to display grids, maps, implications; to set options; and to manage hierarchies of experts, cases, elements, and traits. Even an experienced knowledge engineer may be puzzled as to which technique to attempt next. The Dialog Manager examines heuristics to identify a set of alternatives for the given state of ----CHANGING DIALOG-MANAGER OPTIONS ------[DM]Now changing Dialog Manager options to Assist since you are an experienced AQUINAS user and many knowledge base refinement commands are appropriate here. AQ\*\*OPTIONS DIALOG-MANAGER ASSIST (CR)

Figure 5. The Dialog Manager, still in "automatic" mode, applies an heuristic to automatically switch to "assist" mode when the user is experienced with AQUINAS and there are a large number of operations available.

the knowledge base, and recommends the strategy of greatest potential value, determined by accumulated rule priorities.

We continued to use the Dialog Manager to guide in construction of the DBMS knowledge base. Just as before, the Dialog Manager applies knowledge acquisition heuristics to identify appropriate alternatives and determine the preferred strategy. In this case, the Dialog-Manager finds that the two elements IDMS and TOTAL are highly similar (100% match), indicating the rating grid cannot meaningfully distinguish these two solutions. The Dialog Manager's heuristics specify that analyzing element similarities and adding a trait to discriminate between the two similar elements would be the strategy with the greatest potential payoff to correct the problem. (This heuristic is illustrated later in figure 11). Heuristics dealing with correcting problems detected in the knowledge base are usually weighted more heavily than other command options. When more than one knowledge base problem exists, the one with greatest severity is given priority.

Figure 6 illustrates interaction with the Dialog Manager in "assist" mode. When the Dialog Manager recommends ANALYZE SIMILARITIES ELEMENTS, it pauses until the expert confirms its recommendation with a carriage return or overrules it by erasing the line and entering a new command. The fact that the user confirmed (or overruled) the Dialog Manager's recommendation is recorded on the recent events list and permanent history for analysis by the learning facility.

Next, the Dialog Manager applies an heuristic, based upon complexity of the knowledge base. As shown in Figure 7, the Dialog Manager now recommends

----[DM] Do you want to add elements, add traits, test the knowledge, or correct problems in the knowledge base. I recommend that you correct problems by analyzing similarities elements since there are fewer than 6 traits and there is a high degree of similarity between two elements. Analyze similarities to distinguish them. AQ\*\*ANALYZE SIMILARITIES ELEMENTS (CR) (user confirms)

The two elements IDMS and TOTAL are matched at the 100% level. Can you think of a trait that distinguishes between them? AQ\*\*<u>YES</u> What is that trait?

AQ\*\*GOOD REPORT WRITER What is that trait's opposite? AQ\*\* POOR REPORT WRITER

What is the name of a scale or concept that describes GOOD.REPORT.WRITER/POOR. REPORT WRITER?

AQ\*\* REPORT WRITER

Figure 6. The Dialog Manager, now in "assist" mode, presents an array of command alternatives and suggests ANALYZE SIMILARITIES ELEMENTS. The justification for this selection is a version of a rule in the Dialog Manager. that the expert use cluster analysis to analyze the knowledge base since the rating grid is large with more than 40 grid cells. The number of rating grid cells is just one measure of knowledge base size and complexity used in the Dialog manager heuristics. Since there is also a weak link in the element clustering relationship, the Dialog Manager recommends that the expert SHOW CLUSTERS ELEMENTS to perform a cluster analysis on knowledge base elements. The expert agrees with the recommendation and confirms it with a carriage return. After the DBMS element cluster analysis is graphically displayed (Figure 8), the Dialog Manager proceeds (Figure 7) to suggest

----[DM] Do you want to add elements, add traits, analyze the knowledge, make hierarchies, or test the knowledge? I recommend you analyze the knowledge base by performing a cluster analysis since there are more than 40 cells in the rating grid and there is a weak clustering relationship between element groups. AQ\*\*SHOW CLUSTERS ELEMENTS (CR) (user confirms)

---[DM] Do you want to add elements, add traits, analyze the knowledge, make hierarchies, or test the knowledge? I recommend you MAKE HIERARCHIES by splitting the rating grid on elements because the grid has more than 5 traits and 40 cells and a weak element clustering link. AQ\*\*EDIT SPLIT GRID BY ELEMENTS (CR) (user confirms)

---SPLITTING GRID BY ELEMENTS---Do you want to use the element clusters? AQ\*\*<u>YES</u>

What is a class name for IDMS, TOTAL, SIR, IMS? AQ\*\*HOST LANGUAGE DBMS

What is a class name for EASYTRIEVE, SQL/DS, RIM? AQ\*\* FOURTH GENERATION TYPE

Figure 7. The Dialog Manager, in "assist" mode, recommends SHOW CLUSTER ELEMENTS and EDIT SPLIT GRID BY ELEMENTS. The ustification for this selection is a version of a rule in the Dialog Manager based on knowledge base complexity..



Figure 8. Based on the recommendation of the Dialog Manager, the expert requested SHOW CLUSTERS ELEMENTS for the DBMS knowledge base. The two major clusters are (IDMS, TOTAL, SIR, IMS) and (EASYTRIEVE, SQL/DS, and RIM).

that the rating grid be split by element clusters since there is a clustering link of only 60% between the two groups (IDMS, TOTAL, SIR, and IMS) and (EASYTRIEVE, SQL/DS and RIM). The expert identifies a class name "Host Language DBMS" for the first group and "Fourth Generation DBMS" for EASYTRIEVE, SQL/DS, and RIM. This results in the addition of two new element classes and another level of abstraction n the element hierarchy.

Our expert continued to use the Dialog Manager to assist in the construction of the DBMS knowledge base. *Figure 9* illustrates the hierarchy map for the completed DBMS knowledge base.

#### DISCUSSION

## Strengths and Weaknesses of Approach

Our experience with the Dialog Manager has demonstrated that it is possible to define a knowledge acquisition environment and to formalize knowledge about the transfer of expertise in a rule-based representation. Probably the most valuable result of the implementation of the Dialog Manager is the collection of heuristics for knowledge acquisition and the effective use of Aquinas. A classification scheme is proposed in the following section showing the variety of heuristics that we encountered.

Our research effort focussed on the knowledge acquisition process and improved our understanding of how expertise can be transferred effectively. We were surprised at the number of different strategies practiced by individual experts and knowledge engineers. Our attempt to capture these strategies and formalize this knowledge in the Dialog Manager is an important step in understanding knowledge acquisition.

One disadvantage of the Dialog Manager is that individual experts may (at least initially) disagree with the strategies suggested by the Dialog Manager. It may be that the heuristics used are not appropriate for certain classes of problem or types of idiosyncratic behavior. As the scope of the Dialog Manager's heuristics expands to encompass a variety of strategies, we anticipate that this weakness may be alleviated. A learning capability has been incorporated in the Dialog Manager. Its strengths and weaknesses are discussed later in this section.

We have encouraged users to operate in the "assist" mode since it provides feedback to the Dialog Manager and system developers for both the case when the user agrees with the recommendation of the Dialog Manager or diagrees and overrules the decision. However, some experts object to having to erase the Dialog Manager entry and replace it with their own selection. Users have also criticized the occasionally excessive detail in the Dialog Manager explanations and prompts.



Figure 9. AQUINAS and the Dialog Manager were used to develop the DBMS knowledge base. The completed rhierarchy map consists of 10 DBMS elements, 12 trait pairs, 2 levels of traits, and 3 levels of elements..

22-9

#### A Proposed Classification of Heuristics for Expertise Transfer

It was apparent in our work in developing the Dialog Manager that the heuristics for knowledge acquistion and expertise transfer could be classified. We began to detect certain patterns in the various heuristics for knowledge acquisition. We also observed that the components of these heuristics tended to fall into specific categories. This led to our attempt to define a preliminary classification of heuristics for knowledge acquisition and the transfer of expertise, summarized below.

## Heuristics for Knowledge Acquisition

- Heuristics Based on Temporal Reasoning
  - Heuristics Considering the "Recency" of some event
    - Recent in some absolute sense of time a fixed interval--"within the last 10 minutes"
    - Recent in a "relative" sense --"within the last 10 commands entered"
    - Recency may have a different definition for different experts
  - Heuristics Concerned with a Specific Order of Events (may include functional dependencies)

- whether an event occurred prior to another event
- whether a specific sequence of multiple events occurred in some predefined order
- events which must automatically trigger the execution of some other command
- Heuristics based upon user preferences
  - Individual Preferences
    - Options for interaction selected in the past
      - -degree of interactive assistance from [DM]
      - level of [DM] explanatory detail
    - Patterns of command sequences repeated frequently by a user
      - Repeated patterns in which the user overrules a specific Dialog Manager command with another
      - Repeated past actions of expert to confirm D/M's suggestions
  - User Group or Categorical Preferences
    - Inherited Preference Characteristics
    - Default options for User Group
- Heuristics related to the *complexity* of the knowledge base
  - Number of elements in rating grid
  - Number of traits in grid
  - The number of rating grids
  - The number of experts developing the knowledge base
  - The number of cases considered
  - The number of implications between traits
  - The complexity of the hierarchies
  - Presence of multiple variable types

- Heuristics based upon the Problem Domain
  - Structured Selection
  - Constructive Problems
  - Qualitative Causal Models for Diagnostic Problems
- Heuristics dependent upon *current* state
  - Current state of interaction between user and Aquinas
  - Current state of knowledge base
  - Current state of Dialog Manager
  - Current state of user history
- Heuristics based upon knowledge base problems
  - Heuristics derived from implication analysis, ambiguous relationships in kb
  - Heuristics derived from similarity analysis measures
    - High similarity between traits
    - High similarity between elements
  - Heuristics derived from anomolies in solution and trait hierarchies
- Heuristics Based upon Expertise or Experience of User / User Group
  - Number of knowledge bases created
  - Number of Aquinas sessions performed
  - Number of unique Aquinas commands used.
  - Complexity of knowledge bases
    - Average size rating grid
    - Maximum size rating grid
    - Average number of rating grids
    - Maximum number of abstraction levels
  - Number of times Aquinas used with Dialog Manager off
  - Level of prompt detail requested
- Heuristics for *Prioritization* of Command Alternatives

- Heuristics based on past problemsolving performance (measured by rank correlation with expert)
- Heuristics based upon severity of some problem in the knowledge base
- Heuristics based upon expected benefits of command
- Default heuristics (to be applied when no others are appropriate)

#### Examples of Dialog Manager Heuristics Based on Complexity of Knowledge Base

If number-of-implications > 0 and number-of-traits / number-of-implications < .2

then recommendation =

(ANALYZE IMPLICATIONS ALL) because

*"there is a high number of implications in proportion to the number of traits in the grid."* 

if number-of-implications > 0 and number-of-implications / number-of-gridcells > .833

then recommendation =

(ANALYZE IMPLICATIONS ALL)

"there is a high number of implications in proportion to the size of the rating grid."

Figure 10. Two examples of Dialog-Manager heuristics based on the current state and complexity of the knowledge base.

One important class of heuristics is based on knowledge base complexity. Dialog Manager rules manage hierarchies of experts, cases, elements, and traits. One rule recommends that a rating grid be split into two or more grids when it is large and a cluster relationship between elements or traits exists. Another rule suggests that one expert's rating grid should be subordinated to that of another expert. Other rules attempt to simplify complex implicational relationships. Figure 10 illustrates examples of these heuristics.

### Examples of Dialog Manager Heuristics Based on Problems in Knowledge Base

If fill-in-rating-grid-complete and number-of-ambiguities > 0 then recommendation = (ANALYZE IMPLICATIONS AMBIGUITIES) because

"there are inconsistent or ambiguous implications ir

- ambiguous implications in the knowledge base"
- if fill-in-rating-grid-complete and number-of-elements < 6 and number-of-traits > 3 and element-similarity > .90
- then recommendation = (ANALYZE SIMILARITIES ELEMENTS)

because

"there is a high degree of similarity between two elements in the knowledge base. Analyze similarities to distinguish them."

- if number-of-elements < 6 and number-of-traits > 3 and fill-in-rating-grid-complete and average-element-similarity. >70
- then recommendation = (ANALYZE SIMILARITIES ELEMENTS)

because "the average similarity between elements in the knowledge base is > 70 percent. Analyze similarities to distinguish the elements."

Figure 11. A number of Dialog-Manager heuristics deal with *knowledge base problems*. The Dialog Manager will recommend an action to correct the most serious problem.

Another useful class of heuristics illustrated in *Figure 11* stems from potential "problems" in the knowledge base: a high similarity between two traits or elements, high average similarity, ambiguities, too many or too few implications, etc.

Role of Learning in the Dialog Manager

[DM] Expand grid by adding new solutions
AQ**EDIT ADD ELEMENTS (CR)
Please enter a list of solution elements.
AQ**ADABAS
AQ**(CR)
[DM] Now display map to show changes.
AQ**SHOW MAP (CR)
[DM] Do you want to add elements, add
traits, fill in ratings, or make hierarchies?
I recommend you FILL.IN RATINGS since you
have added a new solution.
AQ**EDIT FILL.IN RATINGS (expert
AQ**EDIT ADD TRAITS (CR) overrules)
[Dati Funeral said by adding new solutions
[DWJ Expand grid by adding new solutions
AQ <sup>**</sup> EDITADD ELEMENTS (CR)
A A the sector of the solution elements.
AQ***(CR)
AQ <sup>***</sup> (CR)
-[DM] Now display map to snow changes.
-[DM] Do you want to add elements, add
traits, fill in ratings, or make hierarchies?
I recommend you FILL.IN RATINGS since you
have added a new solution.
AQ**EDIT FILL.IN RATINGS (expert
AQ**EDIT ADD TRAITS (CR) overrules)
[DM]I notice that you repeatedly follow a
pattern of EDIT ADD ELEMENTS, SHOW MAP,
and EDIT ADD TRAITS. Will you follow this
sequence most of the time?
AQ** <u>YES</u>
[DM]Adding new heuristic for expert ATN.
Please explain why you do this.
AQ**I PREFER TO ADD ONE ELEMENT AND
_ ONE TRAIT AT A TIME .
(Explanation is saved for analysis by Dialog
Manager developers to extract non-
idiosyncratic heuristics)
Figure 12. An example of <i>learning</i> in the Dialog

Manager. The user has repeated a pattern of adding an element, displaying the rating grid, and overruling the Dialog Manager's command suggestion to add a trait instead. The Dialog Manager detects this pattern, and once the user confirms it, constructs a new heuristic for knowledge acquisition for this expert.

The Dialog Manager is equipped with a The data learning mechanism. structures used to record interaction between the Dialog Manager and the Aquinas user (the recent events list, the permanent history) provide a wealth of information about how Aquinas is used to construct knowledge bases. Whenever the user enters a command sequence in "off" mode or confirms or overrules a Dialog Manager suggestion in "assist" mode, the learning mechanism records this fact and scans the data structures looking for similar patterns that have appeared in the past. Detected patterns of user preferences are stored in the user profile, or may be used to create new heuristics applicable to the individual user or a more general class of users.

The Dialog Manager *learning capability* is illustrated in *figures 12* and 13. Our Database Management System expert is expanding the DBMS rating grid by adding elements, using the Dialog Manager's "assist" mode. The expert follows the suggestions to add an element and display the hierarchy map, but overrides the Dialog Manager's recommendation to fill in ratings and chooses to add a new trait instead. Later in the session, the expert repeats this same pattern. The Dialog Manager's learning mechanism observes the repetition in command sequences, noting that in each case, the expert has overruled EDIT FILL IN RATINGS with EDIT ADD TRAITS. The Dialog Manager points out this pattern and the expert confirms a preference for this command sequence. A new heuristic is created for this individual expert, ATN, to suggest EDIT ADD TRAITS after adding an element and displaying the hierarchy map. This is an example of the Dialog Manager's learning of idiosyncratic heuristics based on individual preference patterns.

The Dialog Manager also is capable of learning heuristics which apply to a more general class of users. *Figure 13* illustrates the learning of a nonidiosyncratic pattern based upon observations of behavior of a group of users. In this case, the Dialog Manager

ADDING AN ELEMENT ---- DISPLAYING RATING GRID---[DM]Now display rating grid to show changes. AQ\*\*SHOW UPDATE GRID (CR) --DISPLAYING IMPLICATIONS---[DM]Now display implications to show changes. AQ\*\*SHOW UPDATE IMPLICATIONS (CR) --DISPLAYING RULES---[DM]Now display rules to show changes. AQ\*\*SHOW UPDATE RULES AQ\*\*ANALYZE TEST CONSULTATION (CR) (expert overrules) [DM] detects pattern occurring earlier. --- [DM] I notice that you repeatedly follow a pattern of SHOW UPDATE GRID, SHOW UPDATE IMPLICATIONS, but not SHOW UPDATE RULES. You TEST CONSULTATION instead. Will you follow this sequence most of the time? AQ\*\*YES ---[DM] Adding a new heuristic for expert JOHN. Please enter a short description of why you do this. AQ\*\*TO REDUCE SCREEN OUTPUT. (Explanation is saved for analysis by Dialog Manager developers to extract nonidiosyncratic heuristics) ----[DM] I notice that other members of class Aguinas-developers (JEFF, DAVE, CATHY) follow this pattern. Does this heuristic apply to all Aquinas-developers? AQ\*\*YES ---[DM] Adding new heuristic for Aquinasdevelopers. Figure 13. An example of *learning* in the Dialog

Manager in which a non-idiosyncratic pattern is observed to apply to a class of users: the "AQUINAS system developers". A new heuristic will be added to the Dialog Manager to direct the dialog for this user class. notes that John, a member of the class Aquinas System Developers, follows a pattern of adding an element, displaying the rating grid, and implications, but overrules the Dialog Manager's suggestion to display the rules. As in the previous example, the Dialog Manager adds a heuristic for the individual user, but it also is aware that other Aquinas system developers, Jeff, Dave, and Cathy also follow this pattern. The user confirms that this is a non-idiosyncratic heuristic which applies to the entire user class of Aquinas System Developers.

In both examples of learning (and in any case in which the expert overrules the Dialog Manager), an explanation of why the particular pattern was entered is sought. This textual explanation is stored for later analysis by the developers of the Dialog Manager as a further source of non-idiosyncratic heuristics. In addition, if the Dialog Manager suspects a reason why it was overruled in "assist" mode, it will also display its hypotheses and ask the user to confirm whether that was a reason. For example,

"Did you overrule my recommendation to ANALYZE SIMILARITIES TRAITS because this did not result in improved performance when you did this earlier?"

This proposed explanation is stored for later analysis along with the user's response.

We recognize the weakness in this purely statistical and pattern recognition-based approach, but we feel that it offers promise as a basis for further work. Our intent is for the Dialog Manager to be able to "learn" broader concepts such as how experts organize knowledge hierarchically and how and when the expert should move to a more general or more specific level of abstraction.

#### **ISSUES AND FUTURE WORK**

The implementation of the Dialog Manager and the experience of experts and knowledge engineers who have used Aquinas and the Dialog Manager to develop prototype expert systems have raised a number of issues. We are only beginning to understand how expertise can effectively be transferred: the techniques currently used by experts or knowledge engineers using an automated system such as Aquinas, their underlying heuristics, and whether there are more effective strategies for knowledge acquisition.

We expect to learn more about heuristics for effective transfer of expertise as more experts and knowledge engineers have the opportunity to use Aquinas with the Dialog Manager. We will observe and interview Aquinas users of varying backgrounds and experience in the development of expert systems. We will also examine the information recorded by the Dialog Manager when it adapts existing heuristics to derive new ones.

In addition, we are planning an experiment in which experts will use Aquinas without the Dialog Manager for entry of new knowledge bases, and then repeat the knowledge base entry exercise with the Dialog Manager automatically directing or assisting the process. Transcripts for knowledge acquisition both with and without the Dialog Manager can then be compared and discrepancies discussed with the expert to *identify where Dialog* Manager heuristics could be improved.

The *learning capability* of the Dialog Manager is an area where we are focussing attention. We recognize that the current approach, relying on statistical analysis of the command history and user profile files is an oversimplified view, but we feel that it is a promising first step. Our objective is that the Dialog Manager improve its performance through adaptive learning. As the scope of Aquinas is expanded, the usage of the tool will become increasingly demanding. The recent implementation of Aquinas (Boose, Bradshaw, 1986) allows hierarchically structured cases, experts, solution elements, and traits. This gives rise to a new set of questions to be considered in formulating *heuristics for this complex* environment: for example, when should the rating grid be split into two or more grids, and what is the best way to structure the knowledge? The novice user of Aquinas may find the multiple windows, rating grids, and implication graphs confusing. The need for guidance, explanation, and tutoring becomes more critical as the knowledge acquisition environment becomes more complex.

One of the greatest challenges for Aquinas is **acquiring underlying** concepts in the problem domain. The ROGET system (Bennett, 1983) is a knowledge acquisition tool which aids in conceptualization of knowledge bases. ROGET is aware of the organization of existing knowledge bases in the medical domain, and attempts to structure a new knowledge base using categories common to related expert systems (for example, determined causes, laboratory tests, observed signs). Like ROGET, the Aquinas Dialog Manager carries on a dialog with the expert about the relationships between objects in the problem domain (traits, elements, etc). Both ROGET and Aquinas are concerned with acquiring an expert's conceptual structure . However, ROGET's success may in part be due to the fact that it deals with the medical domain in which categories of advice and evidence are applicable to a whole family of expert systems. Aquinas is not restricted to a specific domain, however broad, and must handle a wider variety of classes of problems.

We also anticipate that the requirements for guidance from the Dialog Manager will increase as the capability is added to acquire knowledge for problems domains which do not fit the structured selection paradigm. We plan to extend Aquinas to handle knowledge acquisition for constructive problem-solving such as in the SALT system (Marcus, McDermott, and Wang, 1985) and to represent domain models of qualitative causal relations in acquiring diagnostic knowledge as in the MORE system (Kahn, Nowlan, and McDermott, 1985) and its successor MOLE (Eshelman and McDermott, 1986).

Another issue to be resolved is *how the* Dialog Manager should prioritize alternative strategies when more than one is justified. It must consider the severity of the problems requiring correction versus the expected benefits of a modification. In our earlier example, (Figure 6) the Dialog Manager chose to correct the element similarity because it was the only current problem in the knowledge base. However, if there had also been an ambiguous implication, the Dialog Manager would have applied the first heuristic illustrated in *Figure 11* to correct the ambiguous relationship because of the severity of the condition (rules cannot be generated, and the knowledge base cannot be tested). Various schemes for evaluating the relative seriousness of problems and predicted success of alternatives will be attempted. A future issue is how do we resolve conflicts between problems of similar severity? How do we determine the command sequences leading to the highest payoff?

Also at issue is the **role of temporal reasoning** in heuristics for knowledge acquisition. Future work will explore the effects of absolute versus relative differences in time, and the concept of "recency" in performing operations in a time-related sequence of commands. For example, if the knowledge has just been tested, and two new traits are added, the user may wish to test the knowledge again even if it was done recently.

An open question is when the Dialog Manager should consider user preferences and past successes to select or rule out a strategy. As an example, the Dialog Manager may decide to suggest that the user volunteer data to expand the rating grid (additional solution elements, trait pairs, or even cases). This activity is appropriate if no data has been volunteered earlier or if many intervening operations have occurred since the grid was expanded. But the Dialog Manager must also consider user preferences and past performance. For example, when the user consistently answers "no" to prompts such as "Can you think of a trait which discriminates between IDMS and TOTAL?", then the Dialog Manager should avoid repeating this request.

A final research objective is to incorporate *user models* in the Aquinas system. Recent attention has been given to user modelling for tailoring system responses, explanation facilities and tutoring for the user, (Sleeman, Appelt, Konolige, Rich, Sridharan, and Swartout, 1985). The Dialog Manager already maintains user profiles and its learning facility is able to make inferences based upon user performance. We will extend this concept to define and model the knowledge, intentions. goals, roles, activities, and communications protocols of Aquinas users, including the domain expert, knowledge engineer, and Aquinas system developer.

User models will be critical if the Dialog Manager is to be used as a tutorial system to teach knowledge engineers and experts how to effectively transfer domain expertise. Woolf and McDonald specify four components in an effective tutoring system: the subject area (knowledge acquisition); the student model (the novice Aquinas user); teaching strategies; and communication strategies. We anticipate that **new** heuristics for teaching and communication must be developed for the Dialog Manager to become an

effective tutoring system. (Woolf and McDonald, 1984; Clancey, 1985). Additional consideration will be given to the interrelationship between the level of user expertise, complexity of the concepts and heuristics, the importance of the concept, and the level of detail in explanations (Wallis and Shortliffe, 1985) to customize the Dialog Manager's explanations for its use as a tutorial system.

#### ACKNOWLEDGEMENTS

The authors wish to thank Roger Beeman, Miroslav Benda, Jeff Bradshaw, Jackson Brown, Art Nagai, Dave Shema, Lisle Tinglof-Boose, and Bruce Wilson for their contributions and support. Aquinas and the Dialog Manager were developed at the Boeing Advanced Technology Center of Boeing Computer Services in Seattle, Washington.

#### REFERENCES

Bennett, J.S., "ROGET: A Knowledgebased Consultant for Acquiring the Conceptual Structure of an Expert System", Report No. HPP-83-24, Computer Sciences Department, Stanford University, 1983.

Boose, J.H., Expertise Transfer for Expert System Design, New York: Elsevier, 1986.

Boose, J. H., "A Knowledge Acquisition Program for Expert Systems Based On Personal Construct Psychology", Int. Journal Man-Machine Studies, 23,(1985).

Boose, J.H. and Bradshaw, J.M., "NeoETS: Capturing Expert System Knowledge in Hierarchical Rating Grids", Proceedings of the Second Annual Conference on Expert Systems in Government, McClean, VA, October, 1986.

Clancey, W.J. "Use of MYCIN's Rules for Tutoring", in Buchanan, B. and Shortliffe, E. (eds.) Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Reading, Mass: Addison-Wesley, 1985.

Eshelman, L. and McDermott, J. "MOLE: A Knowledge Acquisition Tool that Uses its Head", in the Proceedings of the Fifth National Conference on Artificial Intelligence, Philadelphia, Pa, 1986.

Gaines, B.R., and Shaw, M.L.G, "New Directions in the Analysis and Interactive Elicitation of Personal Construct Systems" in M.L.G Shaw (ed.), *Recent Advances in Personal Construct Technology*, New York: Academic Press, 1981.

Kahn, G., Nowlan, S., and McDermott, J., "MORE: An Intelligent Knowledge Acquisition Tool", in the Proceedings of the Ninth Joint Conference on Artificial Intelligence, Los Angeles, California, August, 1985.

Kelly, G.A., The Psychology of Personal Constructs, New York: Norton, 1955.

Kitto, C.M., and Boose, J.H., "An Implementation of Knowledge Acquisition Heuristics", Boeing Computer Services Technical Report BCS-G-2010-[36], 1986.

Marcus, S., McDermott, J., and Wang, T. "Knowledge Acquisition for Constructive Systems", in *Proceedings* of the Ninth Joint Conference on Artificial Intelligence, Los Angeles, California, August, 1985.

Shaw, M.L.G., and Gaines, B.R., PLANET: A Computer-Based System for Personal Learning, Analysis, Negotiation and Elicitation Techniques, in J.C. Mancuso and M.L.G. Shaw, (eds), Cognition and Personal Structure: Computer Access and Analysis, Praeger Press, 1986, in press.

Sleeman, D., Appelt, D., Konolige, K., Rich, E., Sridharan, N.S., and Swartout, B, "User Modelling Panel", *Proceedings, Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, California, August, 1985.

Wallis, J.W. and Shortliffe, E.H., "Customized Explanations Using Causal Knowledge", in Buchanan, B. and Shortliffe, E. (eds.) Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Reading, Mass: Addison-Wesley, 1985.

Woolf, B. and McDonald, D.D., "Building a Computer Tutor: Design Issues," *IEEE Computer*, September, 1984.

¢

## **KNACK - Report-Driven Knowledge Acquisition**

## Georg Klinker, Joel Bentolila, Serge Genetet, Michael Grimes, and John McDermott Department of Computer Science Carnegie-Mellon University Pittsburgh, Pennsylvania 15213

### Abstract

This paper describes a knowledge acquisition tool that builds expert systems for evaluating designs of electro-mechanical systems. The tool elicits from experts (1) knowledge in the form of a skeletal report, (2) knowledge about a large collection of report fragments, only some of which will be relevant to any specific report, and (3) knowledge of how to customize the report fragments for a particular application. The tool derives its power from exploiting its understanding of two problem solving methods and of the different roles that knowledge plays in those two methods.<sup>1</sup>

## 1. Introduction

A key issue in developing any expert system is how to update its large and growing knowledge base. A commonly proposed solution is the construction and use of a knowledge acquisition tool, e.g., TEIRESIAS [Davis 82], ETS [Boose 84], MORE [Kahn 85], SALT [Marcus 85], SEAR [vandeBrug 86], MOLE [Eshelman 86]. Such a tool typically interacts with domain experts, organizes the knowledge it acquires, and generates an expert system. A knowledge acquisition tool also can be used to test and maintain the program it generates. The critical feature of such a tool is that a domain expert can use it to update a knowledge base without having to know about the underlying Al technology.

A large knowledge base can be kept maintainable by organizing it according to the different roles that knowledge plays [Chandra 83], [Clancey 83], [Neches 84]. Knowledge roles, the organizational units of the knowledge base, are made explicit by defining a problem solving method. Some work has been done to analyze existing knowledge acquisition tools with respect to the problem solving method each assumes and the roles that knowledge plays [McDermott 86]. We share the hope that a better understanding of different problem solving methods will lead to better knowledge acquisition tools in the near term and to expert systems with broader scope farther down the line.

KNACK is a knowledge acquisition tool that assists an expert in creating expert systems that evaluate the designs of electro-mechanical systems. Each of the expert systems produced by KNACK is called a WRINGER. A WRINGER gathers information about a system's design, points out possible design flaws and makes suggestions to correct and improve the design.

73-0

201

<sup>&</sup>lt;sup>1</sup>This research was sponsored by the Defense Nuclear Agency (DNA) and the Harry Diamond Laboratories (HDL) under contract DNA001-85-C-0027. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DNA or HDL.

KNACK guides a domain expert through the knowledge acquisition session. It asks the expert to create or modify a skeletal report and report fragments which describe the design of some specific class of electro-mechanical systems. KNACK also elicits knowledge of how to customize the report fragments for a particular application. That information is then transformed into a knowledge representation appropriate to evaluate a system design. KNACK helps the user by minimizing the amount of information the expert must provide to define a piece of knowledge and by using heuristics to infer new knowledge from previously acquired knowledge.

Section 2 gives an overview of the design evaluation task for a specific class of electro-mechanical systems. Section 3 describes the capabilities of the WRINGERs generated by KNACK for evaluating the design of such systems. Section 4 is meant to explicate the problem solving methods and knowledge roles employed by the generated expert systems. Section 5 discusses KNACK.

## 2. The Design Evaluation Task

The WRINGERs that we have developed so far do design evaluation within the nuclear hardening domain. Nuclear hardening implies the use of specific engineering design practices to increase the resistance of an electro-mechanical system to the environmental effects generated by a nuclear weapon. Our domain expertise is supplied by individuals who are technical design reviewers for nuclear effects on electro-mechanical systems. The following is a summary of the problems such experts encounter in their work.

Information about the design is usually provided by several different designers. This causes problems when different designers use conflicting terminology or have inconsistent or incomplete views of the overall system design. The evaluator integrates the information coming from different sources in order to understand the overall system design.

The evaluator tries to gather just enough information from the designer to do the evaluation. The designer takes into account an immense number of details about the system and about the environment within which it will function. Since the evaluation is focused along the nuclear hardening dimension, not every detail that was considered by the designer is relevant for the evaluator. An experienced evaluator knows which information to gather and how to elicit that information.

When one of our domain experts evaluates the design of an electro-mechanical system, he really is helping to design it. He evaluates the design for consistency, completeness, and possible design flaws. Although the designer can develop a system that meets all other requirements, he often does not know enough about nuclear hardening to design a hardened system. Therefore, the evaluation must be constructive. The experienced evaluator suggests improvements to the submitted design which would make it more acceptable.

The designer and the evaluator ordinarily go through several cycles of design presentation and design evaluation. Once the design has been evaluated, it is returned to the designer together with criticism and suggestions on how to correct possible design flaws. The designer then comes back to the evaluator with a revised design proposal. This is an iterative process which leads to an acceptable design proposal.

The evaluator also takes into account highly interdependent information. Restricting a system in some way may influence other design decisions. It may constrain future design choices or require revising previous design decisions.

We have organized these problems into two major tasks for an expert system that functions as a

competent evaluator of electro-mechanical system designs: (1) how to gather just the requisite information from the designer to build a useful design description and (2) how to evaluate the design for consistency, completeness and possible flaws in a constructive manner.

## **3. The WRINGERs**

We decided to build our expert system (the first WRINGER) in two consecutive steps, corresponding to the problems involved. First, we built a system that gathered information. The output from a session with this system was a report describing the design of some particular electro-mechanical system. Because these reports contained all of the information necessary to evaluate the design, they were useful in themselves.<sup>2</sup> Second, we extended our initial expert system so that it evaluated the gathered information. An advantage of this stepwise approach was that our domain experts could critique the generated report and, in the process, show us how they evaluate a system design.

Each WRINGER is a member of a family of expert systems in a common domain area. The WRINGERs we have generated so far gather information about electro-mechanical systems, and evaluate this information with respect to nuclear hardening. They also produce, in report form, a description of the system design and a preliminary evaluation with suggestions on how to improve the design.

#### 3.1. Gathering Information

Usually, a team of designers will work together on the design of a system. It is thus likely that several different designers will interact with a WRINGER to give an adequate description of a system design. Different people might use conflicting terminology. A WRINGER discovers whether two designers use different words to describe the same fact by using its knowledge about synonyms. If a designer's answer to a question contains a synonym for a known expression, the WRINGER replaces it with that known expression.

To take into account the different sources of information available to an evaluator, a WRINGER has several ways to gather information: It can elicit the information from the user by (1) asking a question, (2) interpreting a graphical design description, e.g. a drawing of a system's components, (3) asking the designer to fill in the slots of a table or diagram, or (4) asking the user to choose from the items in a menu. A WRINGER can fill in gaps based on information already available to it by (5) directly applying specific domain knowledge, (6) computing numeric values, or (7) referring to a database. Some information gathering strategies will be more suitable than others for gathering a particular piece of information. Using an appropriate strategy makes it easier for the designer to interact with a WRINGER. For example, to give a spatial description of the system it is often more convenient to draw the system than to answer questions. Often more than one way exists to gather a piece of information. When this is the case, a WRINGER decides how to gather it.

An evaluator might have to try several ways to gather a piece of information before he is satisfied with the result. A strategy used to gather information might not be successful or the information received might be insufficient or incomplete. If a way to gather information was unsuccessful, a WRINGER tries another strategy. For example, a WRINGER uses a database when it has to determine the break-down voltage of a diode. If the database contains no entry for that diode, a WRINGER asks the

<sup>&</sup>lt;sup>2</sup>Most design projects require a finished report containing information about general design objectives, tradeoff decisions, analysis and test results, and detailed parts specifications. Those reports are usually valuable even before they are finished. While it is being written, a good report indicates the benefit of continuing a project; it helps to organize the diverse activities within the scope of a large task and to identify potential problem areas within sub-tasks. A good design report contains all the data that an evaluator needs to determine the adequacy of a proposed design.

designer. On the other hand, if the information received by using some strategy was insufficient or incomplete, a WRINGER will try another way to gather that information. For example, it is possible to infer that one of the subsystems of a system with an internal power supply will be a motor generator. This information might not be sufficient. It is likely that the motor generator is not the only subsystem. Therefore a WRINGER will ask for the remaining subsystems. A WRINGER might consult a database in order to determine the break-down voltage and the damage constant of a diode. If the database contains data on the break-down voltage but no entry for the damage constant, the latter has to be determined. It can be computed if the requisite information is available or else the WRINGER can ask for it.

It is often the case that only a fraction of a WRINGER's knowledge is applicable to the evaluation of any particular electro-mechanical system. A WRINGER selects the necessary information about the design of the system in a data-driven manner; the decision to gather a particular piece of information is based on previously gathered information. Generally more than one piece of information can be gathered at any time. The order in which the pieces of information are gathered may be important: an inference might make a question unnecessary, the designer might feel more natural providing information in a certain order, or information gathered in a certain order might reduce the amount of information needed. Accordingly, a WRINGER uses the following heuristics to select which piece of information to gather next:

- Prefer to infer a piece of information whenever possible.
- Prefer to gather a piece of information that allows the designer to feel comfortable with the order of giving this information.
- Prefer to gather the piece of information which is most likely to reduce the subsequent number of necessary pieces of information, that is, select the piece of information to gather next which prunes the space of necessary pieces of information.

The purpose of the first heuristic is simply to avoid asking the designer unnecessary questions.

The goal of the second heuristic is that the information will seem convenient to the designer to provide next. This is accomplished by following the skeletal report, thereby eliciting information for a selected report part only, and by providing a WRINGER with a set of preference rules enabling it to decide which piece of information to gather next, for example, request the names of subsystems before asking about connections between them.

The third heuristic reduces the burden placed on the designer in gathering a piece of information. WRINGERs prefer to elicit a piece of information which is likely to enable inferences. For example, the type and the diameter of a cable might both be important to know. If a WRINGER has to decide whether to ask a question to determine the type or to ask another question to determine the diameter of the cable, it will prefer to ask the former question because it knows that it can infer the diameter once the type is known. Another way for a WRINGER to reduce the effort to gather a piece of information is to try to establish the requisite information enabling inferences with minimal effort. For example, a worst-case value for the damage constant of a semiconductor device can be inferred if the class (transistor, diode, etc.) and family (high-power, low-power, etc.) of the device are known. On the other hand, the damage constant can be computed precisely if class, break-down voltage, capacity of the junction, material (silicon, gallium arsenide, etc.) and geometry (planar, non-planar) are known. In order to determine the damage constant of a semiconductor device, a WRINGER will try to determine the worst-case value first. Therefore it will try to establish information about class and family. If the worst-case value confirms the system design (does not lead to a design flaw during evaluation), further information needed to compute the damage constant does not have to be gathered, even though the computation would have led to a more precise result.

#### 3.2. Evaluating Designs

Before the design of a system is evaluated for design flaws, the information provided by the designer will be checked to ensure that it is correct, consistent and complete. A WRINGER immediately checks newly gathered information for validity. This includes finding out whether input provided by the designer is obviously wrong or merely implausible. In the first case, an answer might be outside of a predefined numeric range or it might not be a member of a predefined complete set of possible answers. In the second case an answer might be flagged as questionable because it is not a member of a predefined incomplete set of possible answers.

A WRINGER checks the gathered information for contradictions. For example, it makes sure that a cable carrying power is not connected to a cable carrying a signal. It also tries to determine whether it has all the information needed to evaluate the system. For example, it checks whether a power source is specified for the system and whether a defined antenna is connected to the rest of the system.

In the case of the WRINGERs we have built so far, the evaluation of electro-mechanical systems focuses on nuclear hardening. They ensure that a system will function in some predefined nuclear environment. A WRINGER's approach to this evaluation is to determine and use worst-case values for its analysis. An assumption is made that worst-case values are rough estimates of the precise values and that it takes less effort to determine worst-case values (in terms of pieces of information to gather) than to find out the precise values. If the worst-case analysis indicates a design flaw, a WRINGER gathers the additional information that allows it to replace the worst-case values with more precise values. If the evaluation still does not result in a hard system, a WRINGER points out possible flaws in the design of the system. It then suggests improvements. For example, if the energy coupled into an interface circuit through a power line exceeds an upper limit, the semiconductor devices of the interface circuit will be damaged. A WRINGER will suggest using a terminal protection device to limit that energy to an acceptable level.

As a result of the evaluation a designer might want to change his description of the system design or the system design itself. On the other hand it might be necessary to gather additional information. A WRINGER gives the designer an opportunity to review and correct the gathered information about the system being designed. It checks whether all the information necessary to evaluate the system is available. If some vital piece of information is missing, a WRINGER tries to elicit this information from the designer. If the designer cannot or does not want to provide the required information, a WRINGER assumes a worst-case value. For example, a WRINGER needs to know the transfer impedance of the solid shield for a cable in order to evaluate the protection provided by this shield. If the designer cannot provide this information, a WRINGER asks for the outer and inner radii and the resistivity of the solid shield in order to compute the transfer impedance. If this information is not available, a WRINGER assumes a worst-case value.

Finally, the description of the system design and the evaluation results are usually documented in some form. A WRINGER presents the gathered information and the results of the evaluation in the form of a report. It uses the skeletal report to determine the structure and the report fragments to determine the content of a report about an actual electro-mechanical system. The report fragments also guide a WRINGER in integrating the gathered information into a report. In some cases the designer will not agree with the results of the evaluation. For this reason a WRINGER allows the designer to add comments to a report.

## 4. The WRINGERs' Problem Solving Methods

This section describes the knowledge a WRINGER uses to perform the tasks reported in section 3. It gives an overview of the problem solving methods the WRINGERs use, along with the identified knowledge roles, and describes the knowledge bases of two implemented WRINGERs.

A problem solving method is knowledge that establishes and controls the sequences of actions required to perform some task. This control knowledge dynamically defines the order in which subtasks have to be solved in order to perform the overall task. It also defines the kind of domain specific knowledge that is applicable within each step. Thus, the problem solving method helps to identify and classify the domain knowledge. It makes the different roles knowledge plays in the design evaluation task explicit and suggests ways to organize the knowledge base according to the knowledge roles. The granularity of the problem solving method is determined by the demand that the knowledge represented by a knowledge role can be applied without further control knowledge, e.g. the order in which that knowledge will be brought to bear does not matter.

Information gathering and evaluation are treated as separate problems. A WRINGER uses a different problem solving method for each of them. Initially, it gathers the information necessary for the evaluation and then evaluates the information. If, as a result of the evaluation, additional information is required, the WRINGER then invokes the method for information gathering again. After that, another evaluation is performed. This iterative process ends when the designer is satisfied with the design.

#### 4.1. A Method For Information Gathering

The problem solving method and the knowledge roles that WRINGERs use to solve the information gathering problem can be summarized as follows:

- 1. Choose an appropriate part (chapter, section or subsection) of the report that is to be constructed [REPORT STRUCTURE]. Identify all the pieces of information which are required for this part of the report [INFORMATION IDENTIFICATION].
- Determine the pieces of information which are appropriate to be elicited now from the set of all identified pieces of information. If no appropriate pieces of information remain to be gathered, continue with step 6.
- 3. Select one piece of information to be elicited from among those appropriate to be gathered [INFORMATION SELECTION].
- 4. Determine a strategy to gather the selected piece of information and apply that strategy [INFORMATION GATHERING]. If the strategy fails, try another one. If no strategy succeeds, continue with step 2; otherwise check the provided piece(s) of information for synonyms [SYNONYM] and errors.
- 5. Make all appropriate inferences based on the gathered piece(s) of information [INFORMATION GATHERING]. Continue with step 2
- 6. If there are more report parts to be constructed, continue with step 1; otherwise, information gathering is tentatively assumed to be complete.

One role for knowledge is in regard to REPORT STRUCTURE; this includes knowledge about which chapters, sections and subsections are part of the report and knowledge about their order. INFORMATION IDENTIFICATION knowledge is used to identify pieces of information which are relevant to each part of the report; determining an initial set of pieces of information to be gathered. The

INFORMATION SELECTION role represents knowledge about how to select a piece of information to be gathered from the set of all identified pieces of information. It organizes knowledge of how to prefer to gather one piece of information over others. The INFORMATION GATHERING role describes the different ways to gather a piece of information (ask a question, interpret a graphical design description, fill in the slots of a table or diagram, choose among the items in a menu, apply specific domain knowledge to inferences, compute numeric values from formulas, refer to a database). Relying on previously elicited information and other pre-defined knowledge, it defines the circumstances in which these techniques can be applied. It also includes instructions about what to expect as a response and what to do with the elicited information. The role SYNONYM is a way to represent knowledge about making a user's answer consistent with the common way of expressing that answer.

#### 4.2. A Method For Evaluation

The problem solving method and the knowledge roles the WRINGERs use to solve the evaluation problem can be summarized as follows:

- 1. Check the gathered information for consistency [CONSISTENCY EVALUATION] and completeness [COMPLETENESS EVALUATION].
- 2. Evaluate the design description for possible design flaws by using a worst-case analysis [DESIGN EVALUATION]. If no indications of flaws are found, go to step 6.
- 3. Evaluate the parts of the design description which showed indications of flaws again, this time using a precise analysis [DESIGN EVALUATION]. If no indications of flaws are found, go to step 6.
- 4. Make constructive suggestions which pieces of information need to be modified or completed [DESIGN FIX]. Gather any missing information<sup>3</sup>. Generate worst-case values for any required pieces of information still missing [DESIGN DEFAULT]. When there is nothing to be modified or to be completed, go to step 6.
- 5. Continue with step 1.
- 6. Integrate the gathered information into the associated report phrases [REPORT PHRASE], assemble the report phrases, include any evaluation messages, and write the report to an output device [REPORT STRUCTURE].

CONSISTENCY EVALUATION knowledge and COMPLETENESS EVALUATION knowledge include, respectively, how to uncover contradictory information and how to detect when information provided by the designer is incomplete. DESIGN EVALUATION knowledge identifies problem cues that are associated with possible flaws in the design of an electro-mechanical system. The DESIGN FIX role includes knowledge about how the design could be improved. DESIGN DEFAULT knowledge is about worst-case values for the necessary pieces of information the designer could not provide. A WRINGER uses REPORT PHRASE knowledge to incorporate the gathered information into the report. REPORT STRUCTURE knowledge is applied to appropriately place the report phrases within the report.

23-6

<sup>3</sup>The method described in section 4.1 for information gathering is used.
#### 4.3. Two Sample WRINGERs

In Fall 1986 KNACK was used to develop two WRINGERs. The first, a PROGRAM PLAN writer, presents a definition of the activities necessary to design, produce, and maintain an electromechanical system capable of withstanding nuclear environments and completing its mission. The program plan is the primary top level report covering all phases of the design project. It describes the comprehensive plan for management, assessment, hardening, implementation, and testing that a contractor will conduct for an electro-mechanical system. It identifies the specific methods to be used for analysis, design, testing and evaluation, explains the design philosophy, rationale behind various decisions, and assumptions underlying the design. It also projects a schedule of activity and completion for each segment of the program plan. Given several well chosen sample reports, it took one person-week to create the PROGRAM PLAN writer with KNACK. The domain dependent knowledge base contains 834 OPS5 rules: 3 rules describe the skeletal report (REPORT STRUCTURE knowledge), 386 rules define the report fragments (REPORT PHRASE knowledge), 351 rules represent different ways to gather information in order to customize the report fragments for a particular application (INFORMATION GATHERING knowledge), 19 rules define synonyms (SYNONYM knowledge), 46 rules identify the information relevant for each part of the report (INFORMATION IDENTIFICATION knowledge), and 29 rules describe which piece of information to gather next (INFORMATION SELECTION knowledge). The average condition part of one of the PROGRAM PLAN WRINGER's rules has 3.4 elements. Each element is a pattern that can be instantiated by an object. On the average the pattern will mention 2.0 of the attributes which define an object. The average action part has 7.2 elements. Each element creates either a new object or modifies an existing object.

The second expert system, a DESIGN PARAMETERS REPORT writer, presents and evaluates a detailed description of an electro-mechanical system. It documents the system description, analysis, design features, and assumptions required to assure the nuclear hardness and survivability of the system with respect to one nuclear environment: electro magnetic pulse (EMP). To evaluate the system, the WRINGER gathers detailed information about the electro-mechanical system design ranging from the level of major components to the level of individual semiconductors. After the gathered information is checked for completeness and consistency, a worst-case analysis is carried out for each interface circuit in the system, determining whether the EMP environment will induce currents above the operating voltage of the interface circuits. This analysis indicates either that the system is sufficiently refractory of the EMP environment or may not be. In the latter case a more detailed screen analysis, and if necessary an even more precise resistive analysis, is conducted to identify inadequacies in the system's response to the EMP environment. When such an inadequacy is pinpointed, the WRINGER suggests possible fixes, all of which are prechecked for adequate strengthening properties in the interface circuits. The basis for the expert system was a single sample report and a series of interactions with our EMP expert. It took three person-weeks to create it with KNACK. The domain dependent knowledge base contains 1444 OPS5 rules: 3 rules represent REPORT STRUCTURE knowledge, 620 rules REPORT PHRASE knowledge, 610 rules INFORMATION GATHERING knowledge, 32 rules SYNONYM knowledge, 47 rules INFORMATION IDENTIFICATION knowledge, and 29 rules INFORMATION SELECTION knowledge. 135 rules evaluate the gathered information (CONSISTENCY EVALUATION, COMPLETENESS EVALUATION, DESIGN EVALUATION, DESIGN FIX, and DESIGN DEFAULT knowledge). The average condition part of one of the DESIGN PARAMETERS REPORT WRINGER's rules has 3.8 elements. On the average each element will mention 2.3 of the attributes which define an object. The average action part has 3.0 elements. Each element creates either a new object or modifies an existing object.

# 5. KNACK

KNACK is a knowledge acquisition tool capable of building WRINGERs, i.e., expert systems that can evaluate electro-mechanical system designs. Its task is to elicit the domain dependent knowledge from the expert. The remaining part of the paper describes how KNACK aids an expert in creating a WRINGER: (1) it elicits knowledge from the expert about a skeletal report, report fragments, and how to customize the report fragments, (2) it uses heuristics to infer additional knowledge.

KNACK uses object-attribute-value tuples and relations as basic elements to represent knowledge. Each object may have multiple attributes. Dependencies between objects are represented by relations. These basic elements, object-attribute-value tuples and relations, are used to build the condition parts and the action parts of OPS5 rules. An OPS5 rule represents a piece of knowledge. The pieces of knowledge are organized into knowledge roles. A knowledge role is described by a corresponding knowledge role template.

### 5.1. Eliciting Knowledge From The Expert

KNACK's knowledge acquisition approach is based on the assumption that an expert can adequately present his knowledge about electro-mechanical design evaluation in the form of a skeletal report and report fragments. It is also assumed that the expert knows how to customize the report fragments for a particular application. The skeletal report and report fragments describe all the information that reports in a given domain might contain. An actual report will be the skeleton fleshed out with some subset of customized report fragments. In detail, the skeletal report defines the outline of an actual report and the order of report fragments within an actual report. A report fragment describes a small, possible piece of an actual report, the conditions under which the piece will be included in the actual report, the fixed text of a report part and the variable information of a report part. The knowledge of how to customize the report fragments for a particular application defines strategies for how to gather the variable information.

KNACK assumes that the expert knows what information is needed, how to evaluate this information and how a designer should present this information (i.e., the expert must have a clear idea of what constitutes an acceptable report about the design of electro-mechanical systems). We think this is a valid presupposition for a variety of evaluation tasks. In general, someone whose job it is to evaluate the work of others must have comprehensive and precise knowledge about that work.

KNACK provides support to the expert in defining that knowledge. The expert uses keywords to indicate chapter, section and subsection headings. From this KNACK determines the skeletal report. The expert then types the text for a report fragment. There are basically two kinds of text that can be included in a WRINGER report. One kind is 'boiler plate' text and it is simply a sentence or paragraph which will appear exactly the way it was formulated by the domain expert, whenever the conditions for including it are satisfied. The other kind of text contains blanks which are filled in with information that is specific to whichever electro-mechanical system is the subject of the WRINGER report. The blanks are given variable names and it is the values for these variables that are collected by a WRINGER during its information gathering. Figure 5-1 illustrates how a report fragment is defined.<sup>4</sup> The example contains the blanks "<V1>" and "<V2>" for variable information. In an actual report these variables would be instantiated; for example: "The power line interface-circuit is protected by a terminal protection device".

<sup>&</sup>lt;sup>4</sup>In this and following figures, the expert's input appears in bold italics; the implementation details (for rules) and the prompts (of KNACK) appear in lowercase and uppercase; the knowledge inferred by KNACK appears in small capitals. Default responses, enclosed by brackets, are used when the user types only a carriage return.

```
REPORT DEFINITION:

chapter: 4 section: 13 subsection: 0 phrase: 1

[

The <V1> interface-circuit is protected by a <V2>.

]
```

#### Figure 5-1: Sample Input For The Report Phrase Knowledge

The expert then defines object/attribute pairs to represent the variable information. In the example in Figure 5-2 "interface-circuit.name" and "protection-device.name" are used to define the variables " $\langle V1 \rangle$ " and " $\langle V2 \rangle$ " of Figure 5-1. KNACK assumes that objects appearing in the same report sentence are related and routinely creates relations between them, asking the expert to name the relation. If the expert does not give a name for it, the relation is not created. "Protects" is the relation name used in the example.

#### Figure 5-2: Sample Input To Define Variables

KNACK guides the expert through the knowledge acquisition process by exploiting a WRINGER's problem solving methods. A WRINGER's problem solving methods represent the knowledge needed to control the task. They are defined so that the domain knowledge, organized in knowledge roles, does not include control knowledge. KNACK takes advantage of this separation of control knowledge and domain knowledge. It presupposes that a WRINGER's problem solving methods are those described in section 4, thus freeing the expert from having to define the control knowledge.

KNACK also exploits a WRINGER's knowledge roles, which are described by templates. The knowledge role templates define implementation details for a piece of knowledge, they define what the expert has to provide for a piece of knowledge, and they define the optional parts of a piece of knowledge. The implementation details ensure that the domain knowledge can be accessed by the control knowledge. Because these details are already defined by a knowledge role template the expert does not have to be concerned with them. The following example of how to define the knowledge role INFORMATION GATHERING points out how KNACK determines what the expert has to provide to define knowledge. Suppose the expert wants to define a question which a WRINGER will ask the designer. KNACK knows from the knowledge role template for INFORMATION GATHERING that the expert must provide text for the question and that the expert has an option of providing possible and default answers. Therefore it prompts the expert for that input. The sample input shown in Figure 5-3 demonstrates how the expert defines a question intended to gather information for one of the slots in the report fragment shown in Figure 5-1 and 5-2.

• <u>:</u> •

vaciable		EVICE NAME 7.				
	L INDICCITOR D	carectanae 1.	14			
represents:						
object:	PROTECTION-DEV	ICE				
attribute:	NAME					
is determined by.:						
[question, gra	ohics, table, m	enu, inference,	formula,	database,	quit] [	QUESTION]:

text.....: Which protection device protects the interface-circuit possible answers.. [ UNKNOWN ]: default answers..: [ UNKNOWN ]:

Figure 5-3: Sample Input For The Question Knowledge

#### 5.2. Inferring Additional Knowledge Using Heuristics

KNACK uses heuristics to infer additional knowledge. The heuristics can be specific to infer additional knowledge for a particular knowledge role or they can be applicable to more than one knowledge role. We now use the example of Figure 5-3 to describe some of these heuristics. Figure 5-4 shows the question rule<sup>5</sup> for the input sample of Figure 5-3 after KNACK extended it by using heuristics:

KNACK uses heuristics to insert conditions and relations between the objects of conditions into the question rule. These conditions and relations define the circumstances in which asking that question is appropriate:

- If KNACK discovers known objects within the text of the question and if a way is known to gather information about these objects, KNACK adds a condition for each object to the question rule. The question text of the example contains the object "interface-circuit". This is a known object and a question strategy was defined to gather information about this object. Thus KNACK added the condition about the existence of an "interfacecircuit" to the rule.
- KNACK adds a template element relation for each previously defined relation between objects in different conditions. In the example, KNACK added the relation "part-of" between the objects "interface-circuit" and "subsystem".

KNACK uses heuristics to make the text of a question more specific. It is easier to answer a specific question because it is more clear what the question is about. Also, it is easier to evaluate specific information than it is to evaluate some general statements:

 If a known object is used in the question text and if a way is known to gather information about this object and this object is related to another object by a "part-of" relation, KNACK includes this relation into the question text and it adds the corresponding condition to the rule. In the example, KNACK encountered the object "interface-circuit". This is a known object and a question strategy was defined to gather information about this object. It is further known that "interface-circuit" is a "part-of" a "subsystem". Therefore KNACK extended the question text with "of the subsystem" and added the corresponding condition about the existence of a "subsystem" to the rule.

<sup>&</sup>lt;sup>5</sup>A question rule consists of two OPS5 rules: one rule to enable the asking of the question and another rule to handle multiple answers. In the case of multiple answers the second rule fires again for each answer and creates the appropriate basic elements for each of the answers.

```
If
         a SUBSYSTEM WITH SOME NAME IS KNOWN
     and a INTERFACE-CIRCUIT WITH SOME NAME IS KNOWN
     and THE INTERFACE-CIRCUIT IS PART-OF THAT SUBSYSTEM
Then ask "Which protection device protects the interface-circuit (INTERFACE-CIRCUIT.NAME)
          OF THE SUBSYSTEM (SUBSYSTEM.NAME)"
     and create the slot PROTECTION-DEVICE.NAME for the answer
     and create the relation PROTECTS between the object PROTECTION DEVICE
                                           and the object INTERFACE-CIRCUIT
(p question-65
    (questioning-in-progress)
    (SUBSYSTEM ^id <id-1> ^NAME {<SUBSYSTEM.NAME><>NIL})
    (INTERFACE-CIRCUIT ^id <id-2> ^NAME {<INTERFACE-CIRCUIT.NAME><>NIL})
    (relation ^name PART-OF ^from-object INTERFACE-CIRCUIT ^from-id <id-2>
        ^to-object SUBSYSTEM ^to-id <id-1>)
   -(question `number q65 ^id-list <id-1> <id-2>)
---
    (bind <question-id> (gint))
    (make question-text ^id <question-id>
        ^text Which protection device protects the interface-circuit <INTERFACE-CIRCUIT.NAME>
              OF THE SUBSYSTEM (SUBSYSTEM.NAME)))
    (make question ^number q65 ^id-list <id-1> <id-2>)
    (make argument-list q65 <question-id> <id-1> <id-2>)
    (make build-obj/att q65 <question-id> <id-1> <id-2>)
)
(p question-obj/att-65
   {(build-obj/att q65 <question-id> <id-1> <id-2>) <cb>}
  ->
    (remove <cb>)
    (bind <o-id-1> (gint))
    (make object ^id <o-id-1> ^pointer <question-id> ^name PROTECTION-DEVICE)
    (make attribute ^pointer <o-id-1> ^name NAME ^value nil)
    (bind <n-id-1> (gint))
    (make attribute ^pointer <n-id-1> ^name name ^value PROTECTS)
    (make attribute ^pointer <n-id-1> ^name from-object ^value PROTECTION-DEVICE)
    (make attribute ^pointer <n-id-1> ^name from-id ^value <o-id-1>)
    (make attribute ^pointer <n-id-1> ^name to-object ^value INTERFACE-CIRCUIT)
    (make attribute ^pointer <n-id-1> ^name to-id ^value <id-2>)
```

### )

#### Figure 5-4: Sample Question Rule Extended By Heuristics

 If a known object is used in the question text and if a way is known to gather information about this object and if this object has a "name" attribute or a "type" attribute, KNACK will include a variable in the question text which will insert the name or type of the known object. The above example now contains the objects: "interface-circuit" and "subsystem". They are known objects and question strategies were defined to gather information about them. Also, a "name" attribute exists for both objects. Thus KNACK made the question text more specific by adding the variables "interface-circuit.name" and "subsystem.name".

KNACK uses heuristics to extend its knowledge about known objects, their attributes, and the relations between objects. It defines new objects, adds new attributes to objects, and creates the necessary relations between objects:

 If the expert uses an object that KNACK does not know, KNACK asks whether it is a synonym for an existing object. If the expert indicates that it is not a synonym for an existing object, KNACK defines the new object.

- If the expert describes a known object with an attribute that KNACK does not know, KNACK asks whether it is a synonym for an existing attribute of the known object. If the expert indicates that it is not a synonym, KNACK defines the new attribute for the known object.
- If the expert mentions known objects in the text of a question and if a way is known to gather information about these objects and if the object of the answer is a different object, KNACK creates relations between the known objects of the question text and the object for the answer. KNACK also inserts these relations into the representation of the question. The expert has to define a name for each of these relations. The example in Figure 5-4 shows that KNACK extended the question rule with an action to create the relation "protects" between the object for the answer "protection-device" and the known object "interface-circuit" which appears in the question text.

# 6. Conclusion

In this paper we described KNACK, a knowledge acquisition tool for building expert systems that evaluate the design of electro-mechanical systems. KNACK's knowledge acquisition approach is based on the assumption that an expert can adequately present his knowledge in the form of a skeletal report and report fragments. The skeletal report provides a framework around which report fragments relevant to the design of a specific electro-mechanical system can be organized. KNACK also elicits knowledge about how to customize the selected report fragments for a particular application. The skeletal report, the report fragments and the knowledge of how to customize the report fragments provide KNACK with all it has to know to present the design of an electromechanical system and to evaluate it. In order to acquire the knowledge necessary to solve the information gathering and the evaluation tasks, KNACK exploits a WRINGER's problem solving methods and knowledge roles to guide the expert through the knowledge acquisition process. KNACK determines what an expert has to provide to define the knowledge. Finally, KNACK uses For example, heuristics are used to define the heuristics to infer additional knowledge. circumstances in which it is appropriate to ask a question, to make the text of a question more specific, and to extend knowledge about objects, attributes and relations between objects.

# Acknowledgments

We would be remiss if we did not mention our co-workers in this project. William Rodi (S-Cubed) made significant contributions. Casey Boyd (CMU) reviewed an earlier draft of this paper. Thomas Flory and Roland Polimadei of Harry Diamond Laboratories (HDL) and Rodney Perala of Electro Magnetic Applications (EMA) served as our domain experts. We would also like to thank Andrej Bevec (HDL), John Northrop (S-Cubed), William Proffer (S-Cubed), and Alex Stewart (HDL) for their support.

# References

[Boose 84]

Boose, J.

Personal construct theory and the transfer of human expertise. In Proceedings of the National Conference on Artificial Intelligence. Austin, Texas, 1984.

[Chandra 83] Chandrasekaran, B. Towards a taxonomy of problem solving types. *Al Magazine* 4(1), 1983.

[Clancey 83]	Clancey, W. The advantages of abstract control knowledge in expert system design. In Proceedings of the 3rd National Conference on Artificial Intelligence. Washington, D.C., 1983.
[Davis 82]	Davis, R. and D. Lenat. Knowledge-Based Systems in Artificial Intelligence. McGraw-Hill, 1982.
[Eshelman 86]	Eshelman, L. and J. McDermott. MOLE: a knowledge acquisition tool that uses its head. In Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, 1986.
[Kahn 85]	Kahn, G., S. Nowlan and J. McDermott. MORE: an intelligent knowledge acquisition tool. In Proceedings of Ninth International Conference on Artificial Intelligence. Los Angeles, California, 1985.
[Marcus 85]	Marcus, S., J. McDermott and T. Wang. Knowledge acquisition for constructive systems. In Proceedings of Ninth International Conference on Artificial Intelligence. Los Angeles, California, 1985.
[McDermott 86]	McDermott, J. Making expert systems explicit. In Proceedings of 10th Congress of the International Federation of Information Processing Societies. Dublin, Ireland, 1986.
[Neches 84]	<ul> <li>Neches, R., W. Swartout, and J. Moore.</li> <li>Enhanced maintenance and explanation of expert systems through explicit models of their development.</li> <li>In Proceedings of IEEE Workshop on Principles of Knowledge-based Systems. Denver, Colorado, 1984.</li> </ul>
[vandeBrug 86]	van de Brug, A., J. Bachant, J. McDermott. The Taming of R1. IEEE Expert 1(3), 1986.

23-13

. •

## GENERALIZATION IN A NOISY ENVIRONMENT : The need to integrate symbolic and numeric techniques in learning

## Yves KODRATOFF, Michel MANAGO Inference and learning group LRI, Univ. Paris-Sud & CNRS, Bât. 490- 91405 ORSAY, France

Jim BLYTHE, Clive SMALLMAN GEC research, West Hanningfield Rd, Great Baddow CHELMSFORD, England - CM2 8NH

Thierry ANDRO COGNITECH, 167 rue du Chevaleret, 75013 PARIS, France

#### ABSTRACT

This paper describes the main principles by which an integration of two approaches to Machine Learning can be performed. The work is done under the ESPRIT contract P1063, the INSTIL project, which is concerned with the application of machine learning to knowledge acquisition for intelligent knowledge based systems. The intended approach is to integrate several different approaches to Machine Learning and apply the composite system to knowledge acquisition for commercially available expert systems.

## I-INTRODUCTION

There are several ways to look at Machine Learning. In <u>Similarity Based Learning</u> (SBL) (see for instance [Michalski 1984; Lebowitz 1986]), one learns from the similarities detected among a set of positive examples and their differences with a set of negative examples. The approach chosen in this project belongs to SBL. A distinction, inside SBL, can be made in terms of "symbolic" versus "numeric" learning. These two approaches differ in their means and their goals.

The mean of the numeric approach is the optimization of some global parameter like <u>entropy</u> in the case of Quinlan's ID3 [Quinlan 1983], or a distance between examples as in Clustering Analysis [Diday 1979]. Its goal is to point out a subset of the descriptors that are the most useful with respect to this optimization.

The mean of the symbolic approach is the optimization of a description function, synthesized from the examples. It is often required that the recognition function be <u>complete</u>, i.e. that it should recognize all positive examples, and <u>consistent</u>, i.e. that it should reject all the negative examples. Its goal is to express some conceptual relationship among the positive examples.

It is a well known fact that the numeric approach is rather efficient and robust relative to noise but generates rules that are not easily understandable by a human expert. On the other hand, the symbolic approach is well adapted to interactions with experts but cannot handle noise very well [Dietterich & Michalski 1983; Fu & Buchanan 1985].

One of our goals is to write a system (INSTIL) that keeps the positive features of both approaches, as described in this paper. This learning system is being built from three learning software which

were developped at GEC and LRI-IA and is used to generate a rule base for the expert system TOM developped at COGNITECH which diagnoses tomato diseases.

## **II - BACKGROUND MATERIAL**

In this section, we describe the existing individual sub-systems (learning software and rule base) which constitute the background material of the INSTIL project. We will describe in section IV how these pieces fit together.

## II.A. - MAGGY

MAGGY is an object oriented generalization algorithm [Kodratoff 1986; Manago 1986]. It outputs conjunctive generalizations of the examples. For a given disease, the left hand side of the rules concluding to the disease are the positive examples while the left-hand side of all the other rules are the negative examples. From a set of experts given production rules which can be possibly given by more than a single expert (these will be refered to as the training examples in the rest of this paper) or from a set of observations made by experts (fully instanciated training examples), MAGGY outputs rules which are more general (fewer rules with more general preconditions).

When the generalization fails to be consistent (it covers at least a negative example), MAGGY tries to use near-misses [Winston 1985] to introduce new intermediary concepts in the language of description or fill up information which may be missing. Although it has been argued that in "real life" you rarely find near misses [Fu 1985], we feel that when the positive examples are close to the negative examples or when the language of description is not very rich, near misses are actually bound to occur.

### **II.A.1.** Matching algorithm

MAGGY uses an object matcher to extract all the similarities between the positivive examples (this has been inspired by the strucural matching algorithm of AGAPE [Kodratoff & al 1983, Vrain & al 1986]) which works in two phases:

- Transferring the differences between the positive examples into variables of generalization

- using the dropping rule [Michalski 1983] on all that differs (atomic formulae as well as bindings of variables)

This allows to separate the deductive generalization steps from the inductive ones [Vrain & al 1986]. We claim that the only inductive rule of generalization is the dropping rule. All the others are deductive. Consider for example the following rules of generalization :

- Introducing a variable in place of a constant [Michalski 1983].

This is deductive as one long as the binding of the variable is not dropped:  $f(A_1,...,A_i,...,A_n) \Leftrightarrow f(A_1,...,x_n,A_n) \& EQUAL(x,A_i)$ 

- Climbing a the generalization tree [Winston 1975; Michalksi 1983]. This is again deductive unless the binding of the variable is dropped (note that formally, a different is to be made between RED1 and RED0 since the first one is a unary predicate while the other is a constant. On the semantic level, these represent the same concept).

 $RED1(A) \Leftrightarrow COLOR-OF(x,A) \& EQUAL(x,RED0)$ 

Although this remark might seem a bit trivial (in fact it is not as it allows to claim that generalization is mainly a deductive process), we feel that this is extremely important if one wanted to integrate Explanation Based Learning techniques [Dejong 1981, Mitchell 1983, Silver 1986] into SBL as one could view 99% of generalization as a deductive proof. This could be useful to learn better

heuristics of generalization. Note that we do not intend to do this in the course of the INSTIL project, but it is one of the research topics at the LRI-IA.

#### **II.A.2.** Knowledge representation

MAGGY uses normalized (binary predicates) first order logic to represent the examples (a representation centered on the conceptual objects). This is a logic with three values : TRUE, FALSE and UNKNOWN.

For instance, there is a REPULSIVE GREEN MOLD ON A RED TOMATO WITH NO YELLOW SPOTS would be represented as [x: <ISA MOLD> <IS REPULSIVE> <IS GREEN> <IS-ON y>] & [y: <ISA TOMATO> <IS RED>] & ¬[z: <ISA SPOT> <IS YELLOW> <IS-ON y>]

MAGGY also uses frames [Minsky 1975, Charniak 1977] to represent background knowledge.We feel that it is an improvement over to the classical taxonomic representation as:

- frames allow to use structural knowledge to represent more than a single relation of generality. We use the AKO (A Kind Of = subset) relationship between frames (*for instance MOLD AKO SYMPTOM*) and the PART-OF relationship (*LEAVES PART-OF PLANT*). While the first one is used in taxonomies, the second is not. Therefore, frames allow to represent relations of generality between composite objects which cannot be represented by taxonomies.

- An object may have more than one father. Hence it is possible to have a conceptual object belonging to several different taxonomies (Although biologists view a TOMATO as being a FRUIT, in gastronomy it is sometime thought of as a VEGETABLE).

- Frames represent prototypical situations and (easy) default reasonning. For instance, if the COLOR slot of the object LEAVES is filled with GREEN, we will not need to write down the color of a specific instance.

- Axioms can be attached to a frame as done, for example, in KRL [Bobrow & al 1977] (a procedure/axiom can be attached to a KRL unit/object). Although we have not yet implemented this feature, we plan on attaching axioms of the domain to objects of a frame.

We will have \$IF-NEEDED triggers to fire axioms such as  $MATURE(x) \Rightarrow RED(x)$  and  $YOUNG(x) \Rightarrow GREEN(x)$  which will be attached to the color slot of the object TOMATO.

- Finally, a theoritical limit of taxonomies is that they require that all the sons exhaust the possibilities of their father and that they are mutually exclusive. While this may hold in an ideal environment, in presence of noise this is unlikely to be the case. Furthermore there are fields where polymorphy must be taken into account.

## II.B - MAIN

Main is a program based on Michalski's A<sup>q</sup> algorithm [Michalski 1983]. It is presented with a set of positive and negative examples, all described in terms of a logic called VL that is capable of representing structures and arbitrary objects. A disjunctive rule for recognising examples of the concept, written in terms of the same representation, is built by repeatedly forming conjunctions that describe subsets of the examples, until all the examples are described. These conjunctions are formed by generating plausible descriptions from combinations of the individual statements present in a chosen "seed" example.

This process is very flexible. The expert can start the process off with half-formed concept descriptions, and by altering the search criteria can build descriptons ranging in form from those that merely discriminate the current examples from counter-examples to those that attempt to characterise the examples in some way. There are three features of this approach which we shall see are important in section IV. They are

1) the vast space of potential decriptions is searched efficiently by a beam search;

- 2) the method is capable of generating disjunctive concepts, and representing structural knowledge
- 3) the method is heavily influenced by the examples that are chosen as "seed" examples.

## II.C. NEDDIE

NEDDIE is a learning system similar to Quinlan's ID3 [Quinlan 1983]. Given a set of examples that may belong to a number of different classes, it produces a decision tree that discriminates between the classes. It has been argued that decision trees are not a natural way to represent such data. Thus, the system has been equipped with a method of producing classification rules from trees [Corlett 1983]. The system has several modifications to unable it to cope with three kinds of uncertainty or noise in the training set:

- Uncertainty in the training examples

Each example is represented as a set of feature vectors, consisting of a top level example vector which may point to other vectors as physical parts of the object. Each slot of the vector represents the value of one of the features used to describe the training set. The feature values may have an order relation, and can be subsets of the integers or the set of real numbers. Uncertainty in the value of a feature can be represented either by the symbol DUNNO or by assigning to it a range of values, a list, or an interval in the case of real numbers attributes. Such examples can affect the building of the tree in a number of different parts of the tree, with the effect scaled to the probability of its occurrence there.

- Uncertain conclusions in the tree

The system has been designed to use, at will, a number of preference criteria for building the tree, including entropy and a chi-squared test of the reliability of an attribute. Many such criteria are strill under experiment. Apart from this, the tree can be terminated at any point if its overall reliability, based on a chi-squared test, falls below a certain value called min-branch-safety [Corlett 1983]. Thus the conclusions of a tree can be "fuzzy" since it may terminate before an exact classification has been reached. In such cases, the method of choosing a class for an unseen object (for instance the probability or majority methods [Quinlan 1986]) is left to the user.

- Uncertainty in the class of an example

Just as any feature of an example may contain an explicit representation of uncertainty, the class to which it belongs can be expressed with uncertainty. Many people have noted (see for example [Quinlan 1986]) that the class attribute is particularly sensitive to noise, and in NEDDIE the user can specify a preferred probability distribution over a number of classes for that example. If such examples are used, the tree can never produce a complete classification for every case (obviously).

The method used to produce rules from a decision tree is also adapted to noise. If the original branch tree to which a node relates made an uncertain classification, then the rule produced is allowed to also. It may be considerably simpler than its tree counterpart, but must contain no less uncertainty in the sense of the expected information gained in successfully using the rule [Shannon & Weaver 1949]. Neddie takes the decision tree that it generates, and separates it into its component rules. These are conjunctions of tests along with their conclusions. Each rule is then simplified as follows.

Each test of the conjunction is experimentally dropped. If the new conclusion based on the examples that satisfy the new rule is weaker than the original one, the test is put back in. Otherwise it is dropped permanently. Thus all tests in a rule that were originally in the tree but not directly associated with that rule 's conclusion are dropped. Also, because of the use of information theory, rules with "fuzzy" conclusions can be handled in the same way.

These rules can be given a "flat" representation, understanding that all the descriptors of their left-hand side must <u>simultaneously</u> take the value TRUE, to allow the recognition of a disease. On the other hand, one might consider that some descriptors are more important or better defined, or more efficient than others. One should then take advantage of this knowledge by checking the better ones first. It also often happens that some descriptors are redundant and must be used as merely confirmatory, while some others are to drive the recognition process. All this belongs to the set of strategies usually painfully programmed in order to improve the functioning of an Expert System. In our approach, we shall generate an order on the descriptors.

#### ILD. TOM

This expert system diagnoses tomato plant diseases, finding the most probable disease given a set of symptoms. Advice is given both in terms of treatment and prevention of recurrence. It is an operational system, used by plant protection technicians, agricultural technicians and students.

Its functioning is based chiefly upon an interactive dialogue between the user and the inference engine. The vocabulary is that used by human experts, allowing for no computer jargon or abbreviated forms.

The system is based on approximately 280 production rules using 200 descriptors with certainty factors. These descriptors were obtained from specialists in phytopathology from the French "Institut National de Recherche Agronomique".

The system is still under improvment due to some limitations of the original knowledge base and to a natural evolution in the expertise domain. These improvments will be carried out in parallel with the INSTIL project. Besides obtaining automatically learned rules as described in this paper, we shall also attempt to use INSTIL as a benchmark for the use of Machine Learning, as an aid for the development of industrial Expert Systems.

## III. MOTIVATIONS OF THE INSTIL PROJECT

One of our goals is to obtain fewer rules to diagnose the diseases based on the rules given by several different experts. A second goal is to learn strategies to be used during consultation (which questions to ask first, what are the relevant intermediary subgoals).

Collecting training examples from several experts is very important since different experts often have different views on how to solve a problem (they even sometime disagree on what the diagnose of some specific diseases ought to be). Hence, we expect the data to give a complete picture of the domain and to be as representative as can be. We also intend in the future to use the system in more classical manner, i.e. generate rules from a large collection of observed data.

#### **III.A.** Handling of noise

In signal processing, noise is defined as "everything that should not have been there" and "everything that is missing". When dealing with knowledge, this definition has to be expanded to "everything that disrupt the environment that we want to learn from and cause incorrect consultations". One of the aims of the INSTIL project is to build a system that can detect and recover from noisy data. In achieving this, some classification of the kinds of noise encountered in multi-experts knowledge acquisition is necessary, and a first approximation is presented here.

## **III.A.1.** Types of noise

### a) - Natural noise

We define it to be the noise that originates from the concept themselves. We assume (in this paragraph) that the descriptors provided by the experts are as complete as we need (the noise due to

an inadequate description space will be studied in the next paragraph). We have identified three kinds of natural noise.

- Some concepts are very hard to notice

For instance, a discriminating feature of the disease "Colletotrichum Coccodes" is the presence of tiny black marks on the roots, less than one millimetre in size, which are hard to see and often missed during consultation..

Note that this noise could come from the conjunction of some descriptors (for instance, it is harder to see a green spot on a green leaf than on a red tomato).

- Two concepts are likely to be confused with one another during consultation.

For example, a brown spot on a tomato fruit often looks like rotting tissue.

While the skilful experts who provide the examples to the system are unlikely to confuse the two, there is a strong chance that the user of the system will. This type of noise originates from the difference of skill between the persons who build the system, and the persons who use it.

- There can be what we have called symbolic noise. By this, we mean that there is a real overlap in the concepts (for example, when does dark grey stop being grey to become black?). In this case, the noise exists in the definitions of the descriptors themselves and even the experts could make a mistake. When symbolic noise is present, it is often the case that experts disagree on the value that is to be given to the noisy descriptor.

- Finally, the domain can be ill-defined meaning that the experts do not yet have the knowledge to correctly diagnose certain diseases (for example he cannot be sure wether it is disease A or disease B). This can sometime be due to the fact that some further analysis is needed but that this data is not available during consultation and hence that it cannot appear in the experts rules. For the sake of simplicity, we will consider that in this case the domain is ill-defined eventhough it is not. For instance, several diseases are caused by different viruses and it is not possible to differenciate among these without doing some analysis in laboratory (this data will never become available to the users of the expert system). We deal with this kind of noise by putting all the different classes of examples in a single class and not attempt to do any further diagnose (we will only diagnose the disease VIRUS and not attempt to find which virus is causing the disease).

#### b) - Expert noise

We assume (in this paragraph) that expert noise is not introduced because of natural noise in the concepts. Expert noise can be:

- Spelling mistakes (trivially handled)

- Giving the wrong value to a descriptor (for other reasons than symbolic noise)

- Forgetting to mention a descriptor (for other reasons than the descriptor is hard to see).

This is often caused by the fact that people often forget to mention what ought to be absent (*eg the leaves must not be crumpled*). Note that this is only the case in learning from experts rules (and not from observations) as they have in their heads knowledge like "whenever there is X, then it is not disease A".

- Describing one disease, and attributing the description to another (our plant pathologist might have studied the fermentation of grapes a bit too closely!).

- Basing a description of one disease on a plant that in fact has several diseases.

- Using redundant descriptors. This frequently occurs when the examples are given by several

24-4.5

different experts. We will see in section IV.A. that this is actually a positive feature.

- Providing a language of description which is not rich enough

This last kind of noise occurs when a richer language of description is needed in order to discriminate two diseases or in order to discriminate the generalizations of examples of diseases. Consider the following:

 $R_1$ : [x: <ISA PLANT> <AGE MATURE>] & [spots1: <ISA SPOT> <NUMBER SEVERAL> <COLOR WHITE> <FORM CONCAVE>] & [fruit1 <ISA FRUIT>] & [face1: <PART-OF fruit1> <IS EXPOSED-TO-SUN>]  $\Rightarrow$  SUNBURN

 $\begin{array}{l} R_2: [x: < ISA \ PLANT > < AGE \ MATURE > ] \& [spots1: < ISA \ SPOT > < NUMBER \ SEVERAL > \\ < COLOR \ YELLOW > < FORM \ CONCAVE > ] \& [fruit1 < ISA \ FRUIT > ] & [face1: < PART-OF \\ fruit1 > < IS \ EXPOSED-TO-SUN > ] \implies SUNBURN \end{array}$ 

By climbing the generalization frame, one could conclude that a caracteristic description of SUNBURN is : there are light colored spots on the face of the fruit exposed to the sun.

 $\begin{array}{l} R_{3}: [x: < ISA \ PLANT > < AGE \ MATURE > ] \& [spots1: < ISA \ SPOT > < NUMBER \ SEVERAL > \\ < COLOR \ GREY-BEIGE > < EVOLUTE-INTO \ spots2 > < FORM \ CONCAVE > ] & [spots2: < ISA \\ SPOTS > < COLOR \ BROWN > ] & [fruit1 < ISA \ FRUIT > ] & [face1: < PART-OF \ fruit1 > < IS \\ EXPOSED-TO-SUN > ] & [leaves1: < ISA \ LEAVES > ] & [symptom1: < ISA \ SYMPTOM > ] \Rightarrow \\ NARCOSIS-OF-FRUIT-EXTREMITY \end{array}$ 

Since grey-beige is also a light color, the negative example shows that the color is light expect grey beige (the language of description should be enriched by introducing a new intermediary concept as done in [Fu & Buchanan 1985]) or that we chosed the wrong color frame, or that the expert forgot to provide some frames where WHITE and YELLOW appear. (one could also learn that the color of the spots is stable in the case of SUN-BURN or that there are no symptoms on the leaves).

Automatically identifying which of them is valid requires a study of close negative examples which differ by only one characteristic (Winston's near misses [Winston 75]). If there are none, we have to ask for some help to the experts.

For example, the expert has to say that the language of description ought to be improved since WHITE and YELLOW belong to the frame TRANSLUCENT while GREY-BEIGE does not. (in the present case, the fact there are no symptoms on leaves is also relevant). The correct generalization will then be : there are translucent spots on the part of the fruits exposed to the sun and no symptoms on the leaves.

## **III.A.2** - Observable effects and treatment of noise

Natural noise is handled by discouraging the use of the noisy descriptors. We keep a symbolic RELIABILITY variable associated with the concepts describing the disease. We pass on this information as a heuristic so that the systems favor the most reliable concepts. The RELIABILITY is currently given by the expert, but we may in future be able to detect unreliable descriptors automatically by analysing incorrect diagnoses. In addition, symbolic noise can be detected when two different experts disagree on the value of a descriptor.

In the rest of this section, we will discuss how expert noise can be detected.

III.A.2.a. Overlapping of positive and negative examples.

We consider a positive example and a negative example to overlap when one is a generalization of the other. Such an event is easily detected as soon as we try to build rules and test them. When this occurs, two different actions can be taken.

- By using default domain specific knowledge we try to remove the confusion between the two. The expert is informed of the automatic completion of the examples. Here is an example of this :

Disease of the tomato "tobacco mozaic virus": "A yellowing and mozaic on the leaves"

Counter-example. Disease of the tomato "potato virus Y": "A yellowing and mozaic on the leaves which are crumpled"

Using default knowledge, we fill up the missing information in the example that the leaves are not crumpled (in normal state) instead of applying the dropping rule on this descriptor.

- When this fails, we go back to the human expert asking him to separate the two. If the example is a generalization of the counter-example by only one characteristic (it is a near miss [Winston 1975]), we suggest that the example is made more specific so as to exclude the counter-example. This can be done by specializing a rule or by adding the negation of a descriptor.

Disease of the tomato "sclerotinia sclerotinium": "Brown or beige alteration of the stem, 5 cm up, with diffuse limits"

Counter-example. Disease of the tomato "dydimella lycopersici": Brown or beige alteration of the stem, 5 cm up, with diffuse limits and wet tissue.

From these two descriptions we propose the solution of adding to the description of the example the descriptor "not wet tissue" which becomes dry tissue (which is not a default value).

If the expert cannot give immediate feedback, the faulty examples are put into a class of their own and the system carries on.

## II.A.2.b. An example contradicts domain knowledge.

This is usually caused by a diagnostic error on the part of the expert. Similarity-based learning systems have been criticised in the past for not making much use of domain knowledge, but it will play a role in the INSTIL system for the detection and treatment of noise.

II.A.2.c. Noise in the examples can break clusters.

In the typical problem domain of a learning system, an element of regularity is often present. This has been exploited in a number of systems to suggest examples which may be affected by noise, eg Metaxa3 [Emde 1986] or AQ15 [Michalski & al 1986]. Using this technique, if one example is strikingly different from the others, it is considered "suspect" and the expert is notified.

It must be stressed, however, that unlike the other two effects of noise listed here, this result could be caused by a perfectly good example thought of by a single expert, and is only to be used as a guide rather than a rule.

## **III. B. Other motivations**

We aim at producing a system that keeps the best of each individual systems. This includes:

- Handling of noise as described in the preceding section

- A powerful representation of knowledge. For instance, making use of structural knowledge and first order logic will allow NEDDIE to ask questions such as "Is there a symptom with a light color?" instead of "Is there the attribute red?" etc...

- Finding an intelligent disjunctive generalization of the experts given rules. As we will see in the next section, it is not always possible to find purely conjunctive generalization. Nevertheless, we are not interested in "stupid" generalization such as the disjunct of all the training examples.

- Generate efficient strategies (this includes finding an ordering of the attributes, intermediary subgoals and so on)

## IV - PROPOSED APPROACH OF THE INSTIL PROJECT

## **IV.A** - Setting up the questionnaires

The descriptors contained in the current version of TOM are significant in the case of expertise on tomatoes, since they come from the expert himself. We shall add to these "tomato issued" descriptors, some others used in the context of other plant pathology. In this accumulative process, we will forget insignificant descriptors only. We consider the redundancy that will almost certainly show up to be a positive feature, enabling us to solve cases not expected by the experts. It will also help in the building of rule strategies as described in section IV.B.3.

Once a set of descriptors is chosen, a questionnaire containing them and the possible diseases will be set up. Several experts will be asked to fill up the questionnaire while being shown real diseases. Given a real case, they will have to make a diagnosis, firstly on the name of the disease, secondly on the associated values of <u>all</u> the descriptors. This method has already been used in [Michalski & Chilauski 1980].

We are very sensitive to the importance of a good questionnaire, and its relations to noisy data, as explained above. Therefore, the questionnaire will never be considered as final. It must be modified depending on the results. It is clear that asking again Human Experts to fill it up, perhaps several times, as we presently plan to do, is a cumbersome process. This shows well why some Explanation Based Learning could be useful in order to bring in modifications to the rules already learned, without the need of filling again the questionnaires. But this is at present a purely far-off research topic.

#### **IV.B** Merging the current systems.

In this section we describe how we intend to merge our current systems to produce a system capable of achieving the goals laid out in the previous section.

### **IV.B.1** A combination of Neddie and Main.

The reason for the success of Neddie and systems like it in the presence of noise is strongly linked to the top-down approach of building a decision tree. Individual examples only have a significant effect at lower levels of the tree, and thus an earlier termination of the tree based on a statistical approach can be effective [Blythe, 1986]. We will use this feature of Neddie's approach in the integrated system, although the actual tree structure will not be used as a representation of knowledge.

A drawback of Neddie, however, is that it cannot represent structural knowledge adequately. An attempt to give Neddie this power of representation without altering its search technique would increase the complexity of the algorithm unreasonably. In Main, on the other hand, we have a system with sufficient power that also performs an efficient search through the space of possible concept descriptions.

Part of the INSTIL system will use Main's techniques to select good tests to be built into a Neddie-like search tree. Thus we preserve Neddie's top-down approach and can also make use of

the structural information that is a feature of our plant pathological domain. This sub-system has been dubbed the *Downwards Evaluating Neddie and New-main Integrated System*, and will be referred to as "DENNIS" in this paper.

## **IV.B.2** - Generating diagnostic rules

These will be generated by merging DENNIS and MAGGY. MAGGY is a system which is well suited to produce conjunctive generalizations. Nevertheless, it is often the case that conjunctive generalizations (even minimal ones) are inconsistent (i.e. too general since there is at least one counter-example which is covered by them). Hence, a disjunctive generalization which will reject every counter-examples is needed. Consider the following:

Let E1, E2 and E3 be three examples, and CE a counter-example:

E1: [x: <ISA FRUIT> <COLOR RED> <SIZE VERY-BIG> <TEXTURE SOFT>] E2: [x: <ISA FRUIT> <COLOR GREEN> <SIZE BIG> <TEXTURE HARD>] E3: [x: <ISA FRUIT> <COLOR GREEN> <SIZE VERY-BIG> <TEXTURE HARD>]

#### CE: [x: <ISA FRUIT> <COLOR RED> <SIZE BIG> <TEXTURE ANY>]

A complete conjunctive generalization of the examples is [x: <ISA FRUIT> <COLOR ANY> <SIZE LARGE> <TEXTURE ANY>]. It is intuitively clear that it is minimal and it covers CE. A less general expression must be found (which will countain disjunctions) which can be: [x: <ISA FRUIT> <COLOR GREEN> <SIZE LARGE> <TEXTURE HARD>] / [x: <ISA FRUIT> <COLOR RED> <SIZE VERY BIG> <TEXTURE SOFT>]. This is actually the same as making two packages, one with E1 in it and one with a conjunctive generalization of E2 and E3. There is an alternative consistent disjunctive generalization which is to make two packages , one with a conjunctive generalization of E1 and E3 and the other with E2 in it. In this case, the discriminating attribute is going to be the SIZE instead of the COLOR.

We plan on using DENNIS to find out a set of attributes that cannot be generalized. We then pass on this information to MAGGY and generalize the individual clusters. By taking the disjunction of the clusters which belong to the same class, we produce an intelligent disjunctive generalization of each class. There will be some heuristics used by DENNIS which could be:

- Minimize the number of disjunctions in the generalization

- Maximize the explicability and homogeneity of the packages of training examples (how well the expert can explain the generalization and how well the training examples are linked to the generalization in each package)

- Minimize the number of unreliable tests made at each level of the decision tree (see section III.A.1.a).

- Favour some expert given attributes.

In the previous example, we would pick the disjunctive generalization which favors COLOR as it is a more reliable attribute than SIZE (there is more symbolic noise between BIG and VERY-BIG) than between RED and GREEN unless the user is color blind).

Presently, DENNIS has not yet been implemented and we have only tried to integrate NEDDIE and MAGGY (by writing some software to turn NEDDIE's knowledge representation into MAGGY's). This quick and simple method works as follow: NEDDIE generate a decision tree and MAGGY finds a generalization of the individual clusters. In the near future, we plan on inter-mixing the systems as described next.

When DENNIS will be used to generate decision trees, it is possible that the search space becomes too immense. If this happens, we will rely on MAGGY to create a window in the space of negative examples and to pass on a smaller set of attributes which DENNIS will use to generate the decision tree. This will be achieved by finding a conjunctive generalization of the positive examples and analyze (criticize) the inconsistencies. The negative examples covered by the generalization will

constitute the window, while the attributes that have been over-generalized (since no negative example was covered at the beginning of the process, it is clear that some attributes have been over generalized) will make the smaller set of attributes. Therefore, by looking at the trace of the conjunctive generalization process, we will explain what went wrong (identify the faulty step) and specialize it by introducing a disjunction. This will allow the system to find an intelligent disjunctive generalization as well as an ordering on the descriptors.

## **IV.B.3.** Generating strategies for the diagnosis of diseases

Two features of a typical diagnostic expert system that must be taken into account when the rules are used are

1) the user may not be able to answer all our questions in a typical consultation (for instance, it is ridiculous to ask a user a question about the state of the leaves when the user does a consultation by using a picture which only shows the fruits and he cannot access any other information). Therefore, the system must be able to find more than one way of making a diagnosis and hence find alternative strategies;

2) there will typically be many rules, so an efficient strategy must be found to access the rules that are applicable to a given situation.

Our approach to the first problem is two-fold: we produce rules that save as much information as possible in the plant description, and we do not require that all of the description be verified before making the conclusion associated with the rule. Thus a rule can be "fired" in many different ways.

In some learning domains, an efficient strategy for firing rules may be provided by the expert in terms of sub-goals for the diagnosis. If given it, we would certainly use this information, but if not, INSTIL can be used to generate partitions of the rule base by common features, allowing us to access the rules efficiently.

## V. CONCLUSION

We have presented the main goals of the INSTIL project, which are to build a learning system capable of generating the required knowledge and strategies for a real-world expert system that is capable of functioning well in the presence of noise. We believe that "evolutive" or "adaptive" Intelligent Knowledge Based Systems (IKBS) will grow out of fundamental research such as this. They will have the ability to manipulate and expand their own knowledge bases whilst allowing for human guidance.

Clearly then advances on the current situation will be made. This includes the merging and extension of existing symbolic and numeric learning techniques, to facilitate IKBS knowledge acquisition, the provision of tools for domain experts to capture and encode their knowledge more easily, and the provision of a working prototype as a methodological basis for future research in IKBS.

#### Acknowledgments

This research is partially sponsored by ESPRIT contract 1063 : INSTIL. LRI research in Machine Learning is also sponsored by PRC & GRECO "Intelligence Artificielle".

## REFERENCES

[Bobrow & al 77] Bobrow, D. G. and Winograd, T. An overview of KRL, a knowledge representation language, Cognitive Science, Vol 1, N° 1, pp. 3-46, 1977.

[Charniak 1977] Charniak, E.

A framed painting: The representation of a common sense knowledge fragment., Cognitive Science, Vol 1, N° 4, pp. 355-394, 1977.

[Corlett 1983] Corlett, R. Explaining induced decision trees, Proc. Expert systems, 1983, pp. 136-142.

[Dejong 1981] Dejong, G. Generalizations Based on Explanations, Proc. 7 th IJCAI, 1981, pp. 67-69.

[Diday 1979] Diday E., Simon J.C. *Clustering Analysis*, in Communication and Cybernetics Digital Pattern Recognition, Fu K. S. ed, Springer Verlag Berlin 1979.

[Dietterich & Michalski 1983] Dietterich, T. G. & Michalski, R. S. A comparative review of selected methods for learning from examples. In Machine Learning : An artificial intelligence approach, Vol. 1, chapter 3, Michalski R. S., Carbonell J. G., Mitchell T. M. eds, Morgan Kaufman 1983.

[Emde 1986] Emde, W. Great flood in the Blocks World, proceedings of EWSL, Université d'Orsay 1986.

[Fu & Buchanan 1985] Fu, L. M. & Buchanan, B. G. Inductive Knowledge Acquisition for Rule-Based Expert Systems. Knowledge System Laboratory, Computer Science department Stanford University 1985.

[Fu 1985] Fu, L. M. Learning object level and meta-level knowledge in expert systems. Phd thesis, Stanford university 1985.

[Kodratoff & al 1983] Kodratoff, Y. & Ganascia J. G. *Improving the generalization step in learning*, in Machine Learning : An artificial intelligence approach, Vol. 2, Michalski R. S., Carbonell J. G., Mitchell T. M. eds, Morgan Kaufmann 1986.

[Kodratoff 1986] Kodratoff, Y. Learning expert knowledge by improving the explanation provided by the system. Proc. of the first International Meeting on Advances in Learning, Les Arcs 1986.

[Lebowitz 1986] Lebowitz, M. Integrated Learning : Controlling Explanation, Cognitive Science 10, 1986.

[Manago 86] Manago, M.

Object Oriented Generalization: A Tool for Improving Knowledge Based Systems. Proc. of the first International Meeting on Advances in Learning, Les Arcs (France) 1986.

[Michalski & Chilauski 1980] Michalski, R. S. and Chilauski, R. L. Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybeandisease diagnosis. Policy Analysis and Information Systems, Vol 4 No 2 June 1980.

[Michalski 1983] Michalski, R.S. Theory and Methodology of Inductive Learning, in Machine Learning, an Artificial Intelligence Approach, Michalski R.S., Carbonell J.G., Mitchell T.M. eds, Tioga Publishing Company 1983.

[Michalski 1984] Michalski R.S.

Inductive Learning as Rule-guided Transformation of Symbolic Descriptions : a Theory and Implementation, in Automatic Program Construction Techniques, Biermann A.W., Guiho G., Kodratoff Y. eds, Macmillan Publishing Company, 1984.

[Michalski & al 1986] Michalski, R. S. & Mozetic, I. & Hong, J. & Lavrac N. *The AQ15 Inductive Learning System: An overview and experiments.* Proceedings of the first International Meeting on Advances in Learning, Les Arcs 1986.

[Minsky 1975] Minsky, M.

A framework for representing knowledge, in The psychology of computer vision, P.H. Winston ed, Mc Graw-Hill, New York 1975.

[Mitchell 1983] Mitchell T.M. Learning and Problem Solving, Proc. IJCAI-83, Karlsruhe 1983, pp. 1139-1151.

## [Quinlan 1983] Quinlan, J. R.,

Learning efficient classification procedures and their application to chess end games, in Machine Learning, An Artificial Intelligence Approach, Michalski R.S., Carbonell J.G., Mitchell T.M. eds, Tioga Publishing Compagny 1983.

#### [Quinlan 1986] Quinlan, J. R.

The effect of noise in concept learning, in Machine Learning : An artificial intelligence approach, Vol. 2, Michalski R. S., Carbonell J. G., Mitchell T. M. eds, Morgan Kaufmann 1986.

[Silver 1986] Silver B., *Precondition Analysis : Learning Control Information* in Machine Learning : An artificial intelligence approach, Vol. 2, Michalski R. S., Carbonell J. G., Mitchell T. M. eds, Morgan Kaufmann 1986

[Shannon & Weaver 1949] Shannon & Weaver The mathematical theory of communication. University of Illinois press, Urbana 1949.

[Vrain & al 1986] Vrain C., Manago M., Ganascia J. G., Kodratoff Y. AGAPE: An algorithm that learn from similarities, Proc. European Working Session on Learning -86, Université d'Orsay Paris-sud 1986.

[Winston 1975] Winston, P.H. Learning Structural Descriptions from Examples, in The Psychology of Computer Vision, Winston, P. H. ed, McGraw-Hill, New York 1975.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# DRAFT

#### **ҮАКҮАК** –

Yet Another Kit for Your Acquisition of Knowledge The knowledge acquisition tool clearly not named by a marketeer.

> Jim Kornell General Research Corporation PO Box 6770 Santa Barbara CA 93111-6770

#### Abstract

In the following, an extended abstract is given for YAKYAK, a knowledge acquisition tool. The abstract is broken into four parts. First, our motivation for yet another knowledge acquisition tool is described. Then, in a section called "Context and Constraints", we offer our understanding of the nature of the knowledge acquisition task. We then describe YAKYAK, and finish by offering an initial assessment.

## 1. Motivation

YAKYAK is an effort to extend our past work in knowledge acquisition systems [1,2,3], while taking advantage of other's ideas to provide capabilities our tools have lacked. As we have used our tools, we have become more aware of their strengths and their weaknesses, and YAKYAK is an attempt to augment those strengths while correcting the failings. Beyond this, we felt that recent work has demonstrated powerful techniques, but the systems demonstrating these techniques were either (a.) academic investigations somewhat remote from applications areas (e.g., CLUSTER/S [4]), (b.) didn't go far enough or weren't broad enough (e.g., ETS [5],

RuleMaster [6]), (c.) contained what we consider to be epistemological errors (MORE [7]), or (d.) too far from completion to deter current work (CYC [8]).

Since knowledge acquisition is both important and facinating, we wanted to try our hand at a system integrating our favorite knowledge acquisition capabilities. YAKYAK is the result.

## 2. Context and Constraints

.We believe the most critical requirement for a knowledge acquisition system is that it let the expert chose and direct an *appropriate* level of representation for the knowledge s/he deems important. In building a knowledge acquisition system, the first problem to solve is:

> How can the expert easily (even unthinkingly) create the most appropriate representation for her or his knowledge?

This approach shortcircuits any debate over causal versus evidential knowledge being best for a particular system. If one accepts Hume, the idea of "cause and effect" has no referent in the real world [9,10]. We see no need to attempt to direct the expert to use one or the other kinds of knowledge. Rather, following Kuipers [11], we see causal and evidential knowledge distinguished by whether the state variables needed to draw conclusions are implicitly or explicitly expressed. YAKYAK employs slightly different knowledge acquisition techniques for each, but calls little of the expert's attention to the difference. A knowledge acquisition system should accept knowledge in whatever form(s) the expert feels it is best expressed.

Two complications should be mentioned. First, strategies of applying knowledge affect approaches to gathering it. For example, one may gather knowledge differently for classification [12,31] and construction [13] problems. The capability of the system to use important knowledge may

be constrained. For example, the expert may have a guiding metaphor, but the system may not be able to use it. So, a knowledge acquisition system shouldn't be unable to accommodate important information because of expected constraints in a planned reasoning system; at the same time, it shouldn't waste time gathering knowledge it can't use.

The second complication is the necessity of matching the system to the behavior of the expert. As much as possible, the system should meet the expert in her or his territory, rather than force the expert to adapt to the system. In hand-crafted knowledge acquisition, technical details of knowledge representation and application are hidden from the expert. This is more difficult in an automated system. The more the system can support and reinforce the expert doing what s/he is good at, and the less it can require the expert to do something s/he feels uncomfortable with, the better.

With the above concerns as the background, we designed YAKYAK to handle three kinds of knowledge:

- Propositional -- objects and the static relationships among them;
- Procedural -- the dynamic relationships and behaviors
   observed in the world; and
- Metaknowledge -- an understanding of the meanings, structures, and limits of the above two classes of knowledge.

Although YAKYAK has reasoning abilities, we do not view YAKYAK as a reasoning system per se. While the knowledge structures it produces can certainly support nonmonotonic reasoning, we have not yet included that capability. We do not deal with reasoning with counterfactuals [14], nor do we deal with circumscription for reasoning about commonsense knowledge [15,16].

#### 3. YAKYAK System Description

There are two aspects of YAKYAK to be discussed: *automated interviewing* to gather knowledge, and the *facilities* for representing knowledge in the expert's terms.

#### A. Automated Interviewing

We want to build a *documented knowledge architecture*. A knowledge architecture encompasses both the structure and content of an expert's knowledge about a domain. Content means specific facts and behaviors within the domain. Structure is the way the pieces fit together. A documented architecture holds not only the operational domain knowledge but the justifications and rationales for the knowledge as well.

In the following, the sequence of the YAKYAK interview is given, with the focus on gathering operational knowledge. At each step, the expert is queried for and encouraged to supply documentation knowledge as well. This can take the form of annotation attached to particular chunks of knowledge, or a higher level discussion attached to knowledge clusters.

The goal is an architecture of *knowledge entities* which reflect the structure and content of the problem domain. A knowledge entity is a grouping of related facts and heuristics, of similar abstraction or detail, which are closely coupled to one another and less closely coupled to the rest of the world.

To begin, YAKYAK asks for a name or brief description of the problem to be solved. It then asks for all of the decisions which might be reached, and records whether the goal is to select a decision or to rank some or all of the possibilities. This becomes the highest level knowledge entity.

YAKYAK then asks if there are any subproblems to be solved in order to make a decision. If there are, the process recurses, and the name of each subprocess and its possible decisions are recorded as the next level of knowledge entities. Connections between subproblems are classified as serial, concurrent, or independent. Serial means that subproblem A must be solved before subproblem B; concurrent means that

subproblems A and B must be solved at the same time; and independent means that the ordering of subproblem solutions makes no difference. This process goes on until the expert is satisfied that an appropriate granularity of knowledge entity has been achieved -- that none is larger than (in Seymour Papert's phrase) "a mind-sized chunk". Particular stress is laid on decomposing subproblems with concurrent relationships, since concurrency suggests that parts of each subproblem may relate to the other(s) serially.

This gives YAKYAK is an initial sketch of the knowledge architecture. The next step is to try to fill in the connections of the various knowledge entities to each other and to the outside world. We've already made a start, in that many of the subproblem decisions are likely to be conditionals in the "parent" knowledge entities. YAKYAK asks the expert for each entity of the nascent architecture whether any of the knowledge entity descriptors from the lower levels should be regarded as conditionals. Completion of the process leaves YAKYAK with a starting idea of the internal coherence of the knowledge architecture.

To get the connection of the various knowledge entities to the outside world, we use a technique stolen from ETS. We present triplets of decisions, and ask for conditions which distinguish two of the presented decisions from the third. The distinguishing feature may already be present in the knowledge entity, being a previously specified subproblem. If it is not, it is added. This process is repeated exhaustively for each knowledge entity. Note that unlike ETS, the relations as described by the expert are taken on faith. We do not build entailment graphs nor do we try to disambiguate differences between the expert's stated and manifested internal models.

Throughout the process of eliciting conditions, whenever the number of conditions with a knowledge entity exceeds eight, we urge the expert to consider whether a further decomposition of the knowledge entity is in order [17].

Last, YAKYAK asks for exception ("unless") conditions for each knowledge entity. While these are logically identical to the other conditions, it seems that posing the question in this way encourages the expert(s) to think of relevant exception conditions which might not otherwise be enunciated.

At this point, YAKYAK has the basic knowledge structure. We want to flesh in more detail. YAKYAK now asks the expert for condition definitions. For each condition in each knowledge entity, the ranges of legal values are sought. If a condition is a decision in a lower level knowledge entity, the condition receives the decision values from that entity. Otherwise, the expert is asked directly. Whether the condition values come from decision values or are freshly defined by the expert, an ordering of values is requested. Some standard orderings are presented (linear, circular, mutual exclusion, unordered), or the expert can supply her/his own. The expert is also asked whether any of the values (or ranges of values) should be marked as typical or atypical.

For external conditions (those requiring information from the world), the expert is queried about data reliability. The expert can specify either particular condition values or ranges as being of higher or lower reliability, or that all values for a particular condition are of higher or lower reliability<sup>\*</sup>.

As a last step in building a knowledge architecture, YAKYAK asks the expert to order the conditions and to rank condition saliencies. Conditions can be marked like relations between subproblems: serial, concurrent, or independent. Thus, order of condition evaluation is under the expert's control. The expert is also asked for a linear ranking of the importance of the conditions in making a decision. The options exist to rank them, to decline to rank them (in which case they are treated as of equal salience, except when salience is temporarily affected by the typicality or atypicality of condition values), or to rank them on salience tiers (giving

In MORE, this is called test conditionalization and test differentiation.

two or more conditions identical rankings).

At this point all of the pieces are in place, at least at a first draft level. Henceforth, the expert can at will revert to any stage in the process to modify the knowledge architecture. Knowledge entities can be added, removed, or changed; conditions can be eliminated, elaborated, or circumscribed; decision possibilities can be increased, reordered, or constricted. An important side effect of the process of describing actions in the domain (described below) will be recognition of faults in the knowledge architecture. YAKYAK wants to encourage as much as possible the quick rectification of errors.

There are four approaches to eliciting the behaviors of each knowledge entity. YAKYAK uses scenario generation, completeness and consistency checking, induction, and direct querying. The expert has the choice of approaches and order of use.

Scenario generation allows a rapid exploration of the space within a knowledge entity. The process is simple: example conditions are presented to the expert, who then specifies one or more decisions (with attendant certainty). The strategy of the scenario generator is to create sets of condition values which reflect either typical or boundary situations. When this has been done for each knowledge entity, the completeness checker can be used to fill out the gaps between the typical and the boundary states for each entity.

The rationale for scenario generation is that experts should spend more time doing (answering specific questions within their area of expertise) than describing (reporting their reasoning processes). In general, experts are recognized by their ability to make correct decisions when faced with real circumstances. We feel that interview time spent doing what the experts are good at satisfies two important goals. First, since experts are usually quite fast at this kind of exercise, a large amount of knowledge can be gathered quickly. Second, and at least as important, having the expert engaged in something s/he is good at reinforces the expert's use of the tool.

Besides its rapidity, scenario generation has two further advantages. It provides support for knowledge acquisition in the common cases where the expert only remembers a particular rule when reminded of it by some unusual set of circumstances. And, when used with the completeness checker, it lets the system protect itself to some extent from letting any "area" within a knowledge entity be too sparsely defined -- this in turn supports better partial match inference [32] and graceful degradation.

The completeness checker can be used independently of the scenario generator, to look for underspecified situations within each knowledge entity. Because some (often many) of the conditions will have ordered values, we can look for potential situations which are in some sense "midway" between two known situations. The consistency checker looks for very similar situations within knowledge entities which generate different decisions. Such situation pairs (or groups) may not in fact be inconsistent; however, the system can call such pairings to the attention of the expert for validation or correction.

Our use of induction for rule generation is discussed elsewhere [3,18], and need only be covered briefly here. In common with other systems [19,20,21], we induce general rules from the (usually) specific examples created using the scenario generator. Such general rules can be created and put in place automatically. YAKYAK prefers to show each induced rule candidate to the expert for acceptance, correction, or rejection. To the extent that this induction is a form of learning, it is similarity-based learning. YAKYAK has no facilities for explanation-based learning [22].

Finally, the expert can simply tell the system the rules. In most domains there are some special cases which might not be picked up using the scenario generator. And, there may be situations which appear possible as combinations of condition values but which in fact are never seen<sup>\*</sup>. For either of these cases, or if the expert simply prefers to tell the system straight out, a facility for direct entry of rules is present.

<sup>\*</sup> The expert can use a special decision -- unrecognizable circumstance

<sup>--</sup> to indicate that a combination of condition values is not allowed.

Once the knowledge architecture is complete, the expert can test the system. S/he can examine the decisions when YAKYAK queries itself via the scenario generator. Or, questions can be posed directly to the system. The normal process of knowledge acquisition will have any and all of the above steps reiterated as necessary, until the expert is satisfied with the performance of the system.

#### B. Facilities for Representing Knowledge

There are three interesting aspects to knowledge representation in YAKYAK:

- 1. segregation of levels of knowledge;
- 2. dealing effectively with uncertainty; and
- knowledge of expectations.

Vertical and horizontal segregation of levels of knowledge corresponds to information hiding in traditional software, and is desirable for the same reasons. Vertical segregation means that different levels of knowledge are distinct. This includes (but isn't limited to) segregating domain and control knowledge. Horizontal segregation means that knowledge clusters on the same level of granularity but with different subjects are separated. Two crucial elements for any knowledge system are comprehensibility and maintainability. Segregation of levels of knowledge makes each knowledge entity easier to understand, since all of its contents have a single referent. And, it makes maintenance easier, by isolating all of the knowledge about a particular object or event within a well-defined entity.

Segregation is achieved by requiring that for any knowledge entity, whatever the number of conditionals and conclusions, conditions and conclusions are disjoint sets; and all elements within a knowledge entity partake of only the allowed conditionals and conclusions. (This implies that while conditionals may be correlated -- orthogonal factors are hard to find in the real world -- they are independent. Non-independent

conditionals should be segregated.)

The architecture of knowledge segregation takes two forms. Embedded invocation is the means of structuring vertical segregation. Embedded invocation is instantiated when the value of a conditional in one knowledge entity is the output of a lower-level knowledge entity. Serial invocation is used to represent serial reasoning: the action of a conclusion in one knowledge entity is the invocation of another.

The concept of information hiding may seem odd for knowledge based systems, where transparency of knowledge is usually considered necessary. However, in YAKYAK it serves a number of purposes. Besides supporting comprehensibility and maintainability, it melds control of reasoning with the structure of the knowledge. This makes control easier to understand, and the consequences of modifying control easier to predict. And, it allows induction to operate in a straightforward way on many different knowledge levels. We feel that the concepts of vertical and horizontal segregation of knowledge give important advantages to knowledge organization and application.

YAKYAK handles three kinds of uncertainty. First, there is input reliability: to what extent are particular input values to be trusted? As indicated above, the expert can assign a reliability index to a condition, or tag individual condition values with reliabilities. Second, there is knowledge reliability: the normal uncertainty techniques of expert systems<sup>\*</sup>. Third, the system itself deals with uncertainty of application of knowledge. Because YAKYAK does not require exhaustive coverage of the subject domain, there will be times when a question is posed to the system for which it has no directly applicable knowledge. Thus, it must deal with partial matching and with matching on inductive closeness of perceived to known situations. In this case, there is system-generated uncertainty. YAKYAK uses Bayesian techniques for

A pleasant side effect of knowledge segregation and the structural nature of control knowledge is that it is equally easy for the expert to express uncertainty about control and domain knowledge.

combining evidence [26].

Experts generally know what to expect within their fields of expertise. YAKYAK uses three techniques to model expectation<sup>\*</sup>.

First, the range of possible values attached to rule clauses can be ordered at the discretion of the expert. This means for example that an expert might want to distinguish between a medium, a hot, and a very hot oven. Here there is a linear ranking of values, and the system can use the knowledge of the linearity to understand that a hot oven is more like a very hot oven than is a medium oven. The system can know how possible values within a condition relate to one another.

Second, rule clauses can be tagged for saliency within a knowledge entity. Having YAKYAK know how conditions within a knowledge entity relate to one another lets the system weight the clauses differently in induction, in scenario generation, and in reasoning<sup>\*\*</sup>.

The third technique is coupled to the second. It is the capability for the expert to specify one or more "typical" states for condition values. A problem with saliency rating is its unresponsiveness to exceptional conditions. Oven temperature may be the most salient condition for baking, but not if the stove is on fire. By letting the expert specify one or more typical values, the system can use deviation from typicality to boost salience. A very atypical value for a usually less important clause can raise the salience of the clause to make it more central in reasoning.

These three elements -- condition value ordering, condition

<sup>\*</sup> For an alternate view of expectation driven reasoning, see Langley et al [27].

<sup>&</sup>quot;This saliency ranking contrasts with the path chosen in MORE, which as indicated above we feel to be in epistomological error. In MORE, clauses are distinguished as symptoms and background conditions. We feel that the symptom/background condition distinction suggests a separation between clusters of associated clauses which is unwarranted. If a "background condition" is at a different level of abstraction than a symptom, it should be treated as part of a separate knowledge entity. If it is not, then it really just another symptom. In either case, we believe that letting the expert assign saliency values gives more freedom in creating an appropriate domain representation.

saliency, and condition value typicality -- let YAKYAK "expect" the typical and recognize the atypical. This capability is one we take for granted in human experts, but is more difficult to find in (non-NL) expert systems.

#### 4. Assessment

At this writing (late April '86) YAKYAK is only partially implemented. A comprehensive assessment will have to wait until implementation is complete and we have tested it in at least three domains.

A partial assessment can be based on what has been implemented: mostly algorithms which were adapted from our previous work on TIMM and KATIE. Those programs are described elsewhere [1,2,3]. Our assessment of those programs can be seen in the features we chose to keep for YAKYAK and the extensions we felt were necessary. The old programs were powerful enough to build useful systems, and were very simple to use. Washington office could use them. We hope, but as yet have nothing to demonstrate, that YAKYAK will be a major improvement on those tools.

The biggest hole in YAKYAK, and the feature which most certainly assures its obsolescence, is its inability to deal with metaphor. Considering the importance of metaphor in thinking [23], this is a huge hole. We view the CYC project [8] has having the right goals, but there's a long interim between CYC's arrival and present problems. We hope YAKYAK is a productive approach for knowledge acquisition starting from a knowledge-poor position.

There are three classes of further criticism which could be levelled at YAKYAK. One regards reasoning, the next some details of knowledge acquisition, and the last the visual display of knowledge.

YAKYAK is not a reasoning system. At the same time, it must have reasoning facilities in order for the domain expert to test the developing knowledge base. It was mentioned above that YAKYAK doesn't do counterfactuals, commonsense reasoning, or nonmonotonic reasoning (although the structure is there to support the latter). It could be added that it

has the structure but doesn't do multiple context reasoning [24]. Our view is that (a.) we've got hard enough problems to solve to achieve what YAKYAK sets out to do, and (b.) we'd be nuts to try to do everything. The limited reasoning capabilities in YAKYAK (exhaustive deductive, abductive, and inductive inference) are our solution to the "ambition halting problem."

The second class of criticism regards knowledge acquisition details. That the system doesn't do explanation based learning (EBL) we regard as a natural effect of the YAKYAK strategy of starting from a knowledge poor position. It could be said that as YAKYAK acquires knowledge, it should be able to alter its tactics. We would agree -- we simply have to limit version 1. It isn't the case that there is a well tested and "standard" set of algorithms for EBL. For YAKYAK to integrate situation based learning (which it has) and EBL would be a new line of research at least as broad as that already required.

A more serious criticism is that YAKYAK doesn't do enough to distinguish the expert's internal model from her/his stated model. One of the most attractive features of ETS is that it attempts to isolate and illuminate areas where the expert's stated and manifested models conflict. By so doing, ETS attempts to move toward the actual rather than the enunciated model. This is undisputedly a desirable goal. One of the most interesting pieces of information to come out of PROSPECTOR [25] was the disparity between what the experts claimed to do and what they actually did. YAKYAK at this time fails to take up the advance made by ETS.

Finally, the visual display of knowledge is weak in YAKYAK. Particularly for showing the structure of the knowledge architecture, visual display is critical [28,29,30]. YAKYAK uses some of the capabilities of the Xerox/Interlisp-D machine on which it was (is being) developed, but does not go far enough. We view this as a fault to be repaired, but not as a disabling fault. That is, we feel that we can test the ideas in YAKYAK prior to providing a really effective user interface. We are open to comments and criticism on this view, but we see the display algorithms as being a separable stream from the knowledge acquisition algorithms. We therefore feel safe in waiting until the core of the knowledge acquisition

system is built before turning our attention to the display.

#### References

- Kornell, J., Embedded Knowledge Acquisition to Simplify Expert Systems Development, 1984 Conference on Intelligent Systems, Oakland University, Rochester MI, April 1984
- [2] Kornell, J., A VAX Tuning Expert Built Using Automated Knowledge Acquisition, First IEEE Conference on Artificial Intelligence Applications, Denver CO, December 1984
- [3] Cooper, D., and Kornell, J., Combining Symbolic and Numeric Methods For Automated Induction, AAAI Workshop on Coupling Symbolic and Numerical Computing in Expert Systems, Bellevue WA, August 1985
- [4] Stepp, R., and Michalski, R., Conceptual Clustering of Structured Objects: A Goal-Oriented Approach, Artificial Intelligence 28, February 1986
- [5] Boose, J., Personal Construct Theory and the Transfer of Human Expertise, AAAI-84, Austin TX, August 1984
- [6] Michie, D., Muggleton, S., Riese, C., Zubrick, S., RuleMaster: A Second Generation Knowledge Engineering Facility, First IEEE Conference on Artificial Intelligence Applications, Denver CO, December 1984
- [7] Kahn, G., Nowlan, S., McDermott, J., Strategies for Knowledge Acquisition, IEEE Transactions on Pattern Analysis and Machine Intelligence, September 1985
- [8] Lenat, D., Prakash, M., Shepherd, M., CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks, AI Magazine, Winter 1986
- [9] Hume, D., A Treatise of Human Nature (Vol.I) (1740)
- [10] Russell, B., <u>A History of Western Philosophy</u>, Simon & Schuster, New York NY, 1945
- [11] Kuipers, B., The Limits of Qualitative Simulation, IJCAI-85, Los Angeles CA, August 1985
- [12] Clancy, W., Classification Problem Solving, AAAI-84, Austin TX, August 1984
- [13] Marcus, S., McDermott, J., Wang, T., Knowledge Acquisition for Constructive Systems, IJCAI-85, Los Angeles CA, August 1985
- [14] Ginsberg, M., Counterfactuals, IJCAI-85, Los Angeles CA, August 1985

- [15] McCarthy, J., Circumscription -- A Form of Nonmonotonic Reasoning, Artificial Intelligence 13, February 1980
- [16] McCarthy, J., Applications of Circumscription to Formalizing Commonsense Knowledge, Artificial Intelligence 28, February 1986
- [17] Miller, G., The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information, *Psychological Review*, Vol.63, pp.81-97, 1956
- [18] Kornell, J., A Satellite With A Good Attitude: An Expert System For Stationkeeping, IEEE Westex '86, Long Beach CA, June 1986
- [19] Michalski, R., Pattern Recognition as Rule-Guided Inductive Inference, IEEE Transactions on Pattern Analysis and Machine Intelligence, July 1980
- [20] Cheng, Y., Fu, K.S., Conceptual Clustering in Knowledge Organization, First IEEE Conference on Artificial Intelligence Applications, Denver CO, December 1984
- [21] Chiu, D., and Wong, A., Synthesizing Knowledge: A Cluster Analysis Approach Using Event Covering, IEEE Transactions on Systems, Man, and Cybernetics, April 1986
- [22] Lebowitz, M., Integrated Learning: Controlling Explanation, Cognitive Science, April-June 1986
- [23] Lakoff, G., Johnson, M., <u>Metaphors We Live By</u>, University of Chicago Press, Chicago IL, 1980
- [24] Barton, G., A Multiple-Context Equality Based Reasoning System, MIT Artificial Intelligence Laboratory Technical Report No.715, Cambridge MA, 1983
- [25] Duda, R., Gaschnig, J., Hart, P., Konolige, K., Reboh, R., Barrett, P., Slocum, J., Development of the PROSPECTOR Consultation System for Mineral Exploration, *Final Report, SRI Projects 5821 and 6415*, SRI International, Menlo Park CA, 1978
- [26] DeGroot, M., Probability and Statistics, Addison-Wesley, Reading MA, 1975
- [27] Langley, P., Bradshaw, G., Simon, H., Data-Driven and Expectation-Driven Discovery of Empirical Laws, CIP Working Paper No.443, Department of Psychology, Carnegie-Mellon University, Pittsburgh PA, 1982
- [28] Norman, D., Draper, S., eds., <u>User Centered System Design</u>, Lawrence Erlbaum Associates, Hillsdale NJ, 1986
- [29] Card, S., Moran, T., Newell, A., <u>The Psychology of Human-Computer Inter-</u> action, Lawrence Erlbaum Associates, Hillsdale NJ, 1983
- [30] Richer, M., Clancy, W., GUIDON-WATCH: A Graphic Interface for Viewing A Knowledge-Based System, Report No. STAN-CS-85-1068, Department of Computer Science, Stanford University, Stanford CA, 1985
- [31] Clancy, W., Heuristic Classification, Report No. STAN-CS-85-1066, Department of Computer Science, Stanford University, Stanford CA, 1985
- [32] Hayes-Roth, F., McDermott, J., Knowledge Acquisition From Structural Descriptions, *IJCAI-77*, Cambridge MA, 1977

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

The Knowledge Acquisition Grid: A Method for Training Knowledge Engineers

Marianne LaFrance Department of Psychology Boston College Chestnut Hill, Massachusetts, 02167

This paper describes the Knowledge Acquisition Grid, newly developed to assist knowledge engineers in the manual transfer of expertise. The <u>Grid</u> is used in a knowledge acquisition training module which itself is part of a larger training program in building expert systems offered by Digital Equipment Corporation. The <u>Grid</u> describes a two-dimensional space in which five forms of expert knowledge and six basic types of interview questions constitute the horizontal and vertical dimensions respectively. Description is given of the rationale, components, strategy, and evaluation of the <u>Grid</u> for training in knowledge acquisition.

Knowledge-based systems require the kind of specialized and extensive know-how usually available only in the heads of a small coterie of experts. To date, the gaining of this expertise has proven to be one of the great challenges in building expert systems. This paper describes one component of a Training Program in Knowledge Acquisition designed to provide knowledge engineers with the conceptual and practical means with which to more effectively conduct the manual transfer of expertise.

#### BACKGROUND

The Training in Knowledge Acquisition is currently part of a larger training program in building expert systems first put into service by Digital Equipment Corporation in late 1982 and now offered several times a year to people both inside and outside the company. Although the original 14week course in building knowledge-based systems covered a range of available AI tools, a significant gap in the curriculum was identified by the end of the first year: there was no material specifically focussed on how to deal with the complexities of acquiring knowledge from the experts.

The initial response was to have the author, a social psychologist, design a 2-day module initially called, "Expert Interviewing". The aim was to introduce knowledge

engineers to in-depth interviewing by drawing on theory and research in psychology, ethnography, nonverbal communication, cognitive science, and the sociology of specialized interviewing. The guiding assumptions for the training were twofold: first, that experts who are in the position to know the most may be the least able to articulate what they know; and second, that available methods could be adapted and developed for the expressed purpose of eliciting this particular kind of <u>tacit</u> <u>knowledge</u>. The module has since been expanded and updated to reflect developments in the field as a whole. Now called "Knowledge Acquisition", the training has become an integral and important part of Digital's training program.

#### Focus on Manual Knowledge Acquisition.

Although the course includes material on automated methods of knowledge acquisition, the overall emphasis is on manual interviewing methods. There are a number of First, manual knowledge reasons for this emphasis. acquisition is the most routinely used and broadly useful method of acquiring expertise. Manual methods can be applied to any knowledge domain. Second, even when knowledge acquisition features automated become practically feasible for some aspects of the process, manual methods will still be called upon to complement and cover content not accessible by purely automated means. Third, manual interviewing methods are uniquely good at taking into account considerable expert to expert and domain to domain differences. A final reason for focusing on manual methods of gathering expertise is that as they are used in an increasing range and variety of settings, they will suggest promising directions for how knowledge acquisition can be effectively automated.

<u>Training</u> <u>Objectives</u>. The complete training module in knowledge acquisition is designed to increase knowledge acquisition effectiveness by exposing participants to a variety of techniques and tools. One of these tools is the Knowledge Acquisition Grid which constitutes the central concept of this report. In addition to gaining familiarity with the structure of the Grid, the rationale for its two dimensions, and suggestions for how it can be used to grasp expertise, the training in knowledge acquisition has a number of other objectives. They include the following:

1) An enhanced appreciation for knowledge acquisition as an integral process in building expert systems,

 An understanding of the qualitative aspects of expertise,

3) Facilitiy in identifying the prevailing myths about knowledge acquisition and in dealing with the realities,

4) Competence in using specialized interviewing techniques to enhance the expert's involvement and the knowledge engineer's comprehension,

5) The ability to recognize different kinds of expert knowledge,

6) A repertoire of different types of questions to access different types of knowledge,

7) Awareness of how knowledge acquisition develops, and

8) Acquaintance with new developments in the automation of knowledge acquisition.

#### NEED FOR TRAINING IN KNOWLEDGE ACQUISITION

Expert systems are valuable to the extent to which they are knowledge rich, but the need for the knowledge has not been matched by effective means for drawing it out. Expert knowledge is not easily captured. The process is time consuming, painstaking and complicated. In fact, knowledge acquisition has been called the critical bottleneck problem in the development of knowledge-based systems (Feigenbaum, 1980).

In acknowledgement of this bottleneck, one response has been to increase efficiency by developing means for automating the transfer of expertise. Another tack has been to recognize that several issues contribute to the bottleneck and that methods must be developed to deal with each of them. A number of the important ones are: 1) lack of understanding about the nature of expertise, 2) poor manual interviewing skills, and 3) a limited repertoire of questioning strategies. Each of these are described briefly below.

#### Nature of Expertise

Knowledge engineers who assume that the problem of knowledge acquisition is essentially quantitative are likely to bypass important know-how. Work in cognitive psychology has shown, for example, that experts differ from novices not only in the amount they know, but also in how their knowledge is represented and bundled (Chi, Glaser and Rees, 1982; Chiesi, Spilich and Voss, 1979, Larkin, McDermott, Simon and Simon, 1980; Milojkovic, 1982).

A conception of know-how based on qualitative aspects has important implications for people concerned with acquiring knowledge from experts. For example, there is need to recognize that expert knowledge comes encapsulated in varying modes rather than an undifferentiated mass of information. The Forms of Knowledge Dimension of the Knowledge Acquisition Grid articulates five distinct forms of expert knowledge, each with its own structure and characteristics and each enclosing a subset of the expert's expertise. These will be described more fully below in the section entitled, "The Knowledge Acquisition Grid".

#### Poor Manual Interviewing Skills

The process of knowledge acquistion usually demands an extended and intense collaboration between the domain expert and the knowledge engineer. To say that this requires interpersonal competence and communication skills severly understates what is involved. Experts and knowledge engineers alike have different experiences and preferred ways of thinking and working. If these are ignored the knowledge engineer is in danger of getting an incomplete or misconstrued knowledge base.

Experts cannot be simply hooked-up and drained of everything they know. An approach that takes account of the personal nature of expertise and the inter-personal quality of acquiring it will have more success than one that gives little weight to these important social psychological issues.

#### Limited Questioning Strategies

Designing more proficient knowledge acquisition depends in part on expanding ways to tap into human expertise. It is insufficient to merely have the patience to ask "why" of an expert four hundred times in a row. A enlarged and more targeted repertoire of questions and probes is required.

For example, recent research in cognitive and social psychology has shown that different types of questions elicit different kinds of answers (Clark, 1985; Loftus, 1975). In other words, the form that a question takes constrains the answer obtained. The Types of Questions Dimension of the Knowledge Acquisition Grid articulates six distinct kinds of questions directed at uncovering expert knowledge, each with its own structure and purpose. These also will be described more fully below in the section entitled, "The Knowledge Acquisition Grid".

In sum, these bottlenecks in knowledge acquisition can be relieved through the application of a manual method that approaches various kinds of expertise with a diversified set of questions.

# THE KNOWLEDGE ACQUISITION GRID

Since expertise apparently resists single category compartmentalization, and since no single question can can elicit all the required information, a multi-modal approach was needed. The Knowledge Acquisition Grid addresses this need by conceptualizing the multiple ways that experts encode what they know, and provides a wide repertoire of question types to gain access to it.

#### Structure of the Knowledge Acquisition Grid

The Knowledge Acquisition Grid shown in Figure 1 organizes expert knowledge and knowledge engineer questions as separate but interacting dimensions. The first dimension represented by the columns in the Grid, describes the <u>Forms of Knowledge</u> in which an expert's know-how is stored. The second dimension, represented by the rows in the Grid, describes the <u>Types of Questions</u> that are available to knowledge engineers to get access to it.

Forms of Knowledge

Question Types

Layouts Stories Scripts Metaphors Rules-of-Thumb

Grand Tour

Cataloging Categories

Ascertaining Attributes

Determining Interconnections

Seeking Advice

Crosschecking

Figure 1: The Knowledge Acquisition Grid

#### Dimension 1: Forms of Knowledge

Expert knowledge is both multiform and tacit. Thus expert system builders need a way to recognize and access material that is distributed in memory and implicit in character.

The Knowledge Acquisition Grid presents five <u>Forms</u> of <u>Knowledge</u>. A brief description of each form is as follows:

1) Layouts. Layouts incorporate the expert's "map" of the task, including an understanding of its boundaries and basic elements. Layouts subsume and give coherence to the expert's facts and heuristics by specifying the goals to which they aim, and the criteria used to characterize the problem at hand. By getting access to how the expert sets the task and organizes current information in light of prior knowledge, the knowledge engineer is able to determine the problem type and its scope.

2) Stories. Stories represent the classic cases and typical examples culled from the expert's long experience with the problem domain. Stories can be of a number of types in addition to the now familiar Talking Aloud Protocol (Waterman and Newell, 1971). For example, Explanatory Stories are those in which the expert, seeking to account for some puzzling situation, narrates a set of events that unfold in such a way as to lead up to and produce the phenomenon in question. A Diagnostic Story/Prescriptive Story describes some phenomenon in a way that shows what was wrong and what needed fixing.

3) Scripts. Scripts give the expert's sequential and procedural knowledge of the area. The basic elements of a script are its roles, standard props or objects needed to carried out the actions, a standard sequence of scenes wherein one act enables the next, and some normal results from performing the activity successfully (Abelson, 1981). To know an expert's scripts is to have a flow chart of critical actions, and to be able to understand each action in terms of the prior knowledge required to perform it.

4) Metaphors. Metaphors encapsulate the expert's image of the task which includes its characteristics, constraints and options. Metaphors describe one thing by reference to another apparently dissimilar thing so that the first is understood more completely than if the comparison had not been made. Their advantage is being able to present an idea compactly, so that a collection of characteristics is conveyed in a word or two, which can

later be reconstructed and embellished through probes directed to the expert by the knowledg engineer.

5) Rules-of-thumb. Rules-of-thumb provide the myriad tactics and methods for interpreting and dealing with the vast array of circumstances encountered in carrying out the task. A rule-of-thumb is encapsulates tacit knowledge about which conditions warrant which actions, and about how to gather data on and assess current conditions. Rules-of-thumb are concrete, implementable strategies of minor to moderate scope which can single out and define as issues those specific, limited conditions for which they serve as the most complete strategy.

# Rationale For Multiple Forms of Knowledge

Each of these Forms of Knowledge is a different way in which know-how can be represented in the mind of the expert. Together, the five forms suggest that expertise in a domain is not encoded in the expert's mind in a single bundle, but that varying experiences with the problem lead to different mental representations.

Support for this idea comes from research in cognitive psychology that shows that learning is a segmented and that the various segments are stored process, in parts of the memory different (Anderson, 1980). Psychologists hypothesize that breaking up of information in this way allows people to make better generalizations and more useful predictions. Similarly, Minsky's societyof-mind theory proposes that intelligent action emerges from the interactions of many small systems operating within an evolving overall administrative structure (Minsky, in press).

<u>Advantages of Multiple Forms of Knowledge</u>. The Forms of Knowledge dimension thus sees expertise as made up of a number of smaller subsystems, each with its own character and function. There are, in addition, a number of practical advantages to seeing expert know-how as multimodal. In the first place, the Forms of Knowledge provide a way during knowledge acquisition to catalogue a domain's real complexity. In the spirit of getting a handle on a domain, there is sometimes danger that that which initially does not seem to fit may be left out. The five kinds of knowledge provide the means to tap a domain's particular intricacies. Secondly, the Forms of Knowledge encourages broad coverage. In seeing knowledge as multimodal, the knowledge engineer is more likely to check whether other modes of expressing the expertise will turn up relevant but previously unarticulated know-how.

Third, each Form of Knowledge reflects a different but practical slant on the problem domain. Layouts, Stories, Scripts, Metaphors, and Rules-of-Thumb implicitly prescribe how to deal with different features of the job. In other words, how a problem is laid out or scripted has direct implications what can be done about it.

Next, the five Forms of Knowledge have the advantage in allowing understanding of the expert's domain in his or her own words. This gives the knowledge engineer a way to stay close to concepts of a domain while at the same time having the means to organize them. Finally, the five Forms of Knowledge provide a buffer between acquiring the knowledge from the expert and transforming into a particular representation. This allows the expertise to be understood first on its own terms, a stage which is important for subsequently being able to select appropriate representations for it.

One caveat should be mentioned here: even though expert knowledge is generally multi-modal, not all Forms of Knowledge are evident in all knowledge domains, nor are they used equivalently by all experts. Nevertheless, knowledge engineers should recognize the potential in each area, and not assume that unfamiliar forms are necessarily unproductive ones.

## Dimension 2: Types of Questions

The process of constructing a knowledge base requires the knowledge engineer to have a number of question types. Survey researchers, for example, have shown that using different question formats optimizes the information received. Alternatively, the use of only one type of question severely limits the kind of answers that can be obtained. The Knowledge Acquisition Grid presents five <u>Types of Questions</u>. A brief description of each type with an example is given below:

1) Grand Tour Questions. Grand Tour questions cast a wide net over the domain in order to understand the boundaries as the expert sees them. The material sought includes an overview of the expert's perspective, goals, constraints and options. Sub areas within the expert's domain are comparably pursued by means of Mini-Tour queries. An example of a Grand Tour question is "Could you describe the kinds of things that schedulers do? Please do not edit things out of your description, even things you think may not be important." This particular phrasing of the question is designed to elicit a Layout

description but could as equally be directed at eliciting a Story or Metaphor.

2) Cataloging the Categories. The expected outcome of these questions is an organized taxonomy of the expert's terms and concepts. An example of a Cataloging the Categories Question is "When you gave me an overview of your job, you talked about 'schedulers'. Are there different types of schedulers? Are schedulers a subtype of some other kind of job?" This type of question might follow a Layout response to a Grand Tour Question.

3) Ascertaining the Attributes. These questions aim to discover the distinguishing features and range of possible values of the expert's concepts. An example of an Ascertaining the Attributes Question is "You've described a number of types of scheduling situations that you've encountered. I wonder whether you could now take the first two that you mentioned, and describe some ways in which these two are similar to each other but different from the third example that you gave." As stated, this question is a follow-up to a Stories description.

4) Determining the Interconnections. These are questions directed at uncovering the implicit causal model of the domain. Of particular interest are the relationships and linkages among the expert's concepts. An example of Determining the Interconnections Question is "In describing the routine set of steps for scheduling an order you said that checking the request date occurs before anything else. Why is that the case?" This question is directed at previously obtained Scripts information.

5) Seeking Advice. These questions are designed to reveal the expert's recommendations for how to deal with a variety of conditions such as how to determine current conditions and which conditions warrant which actions. An example of a Seeking Advice Question would be "You've compared scheduling to playing a board game; from your experience with playing board games what advice could you give on doing scheduling?" This type of question would be in response to a previously volunteered Metaphor.

6) Cross-Checking Questions. These questions are designed to validate and examine the limits on previously obtained information. Cross-checking questions actually consist of five subtypes including, the Naive Question, Playing Devil's Advocate, the Hypothetical Situation Question, asking How Sure Are You? and Seeking the Exception. An example of a Naive Question is "Bear with me while I ask what may appear to be a naive question.

Could you tell me why orders need to be scheduled?" An example of a Devil's Advocate Question would be "Let me play devil's advocate in response to your story about the need to set priorities on scheduling multiple orders. What if you didn't set priorities?"

# Rationale For Multiple Question Types

Research indicates that questions variously restrict and direct the kinds of answers that can be obtained. For example, there are two broad types of questions: questions that ask the recipient to provide the questioned item, and second, utterances that require the hearer to simply agree or disagree with the content of the question. Whquestions, those beginning with the words, who, whose, which, what, why and how, ask that the question variable be supplied and hence provide considerable leeway for the recipient to respond. The second type, yes/no questions call on the hearer merely to concur or not with the proposition put forward by the questioner (Woodbury, 1984).

Within the Wh-question category itself, there are differences in the specificity of the information solicited. Questions beginning with what, why, and how are open-ended whereas questions beginning with the words, who, where, when, and which appear to require answers that are as specific as the hearer is able to make them.

Subtle variations in the wording of questions has also been shown to affect how hearers respond to them. For example, research on eye-witness testimony has revealed that witnesses to a car accident report that the car was going significantly faster if the question was phrased, "How fast was the car going when it <u>crashed</u> into the wall?" than if it was put, "How fast was the car going when it <u>ran</u> into the wall?". A comparable rewording in another study showed the same effect. In this case, survey respondents report more headaches if the question is phrased, "Do you get headaches <u>frequently</u> and if so, how often?" than if they are asked instead, "Do you get headaches <u>occasionally</u> and if so, how often?" (Loftus, 1979).

Studies such as these show that replies are significantly effected by how questions are phrased. Other research on questions has focussed on asking whether some question strategies are more effective than others in eliciting full and valid disclosure. The theoretical notion here is that the memory trace is composed of several features and hence the effectiveness of a question or retrieval cue is related to the amount of feature overlap with the event

being reported. In other words, some questions pick up elements that are neglected when other kinds of questions are asked. A second outcome of this research is the recognition that there are probably several retrieval paths to the encoded event, so that information that is not readily accessible by one retrieval cue might be available with a different cue (Tulving, 1974).

The Knowledge Acquisition Grid is based on this theoretical framework. The matrix formed by forms of knowledge and types of questions was designed to increased the feature overlap between the encoding of the expertise by the expert and the subsequent retrieval of it by a knowledge engineer in a knowledge acquisition context. The multiple forms of both knowledge and questions encourage using many retrieval paths.

Advantages to Multiple Question Types. There are then a number of advantages to being able to call upon a number of different question types. The six types of questions in the Grid are designed to be seen as a package, the contents of which are geared to producing total а comprehensive survey of the expert's knowledge. Secondly, the questions enable the knowledge engineer to approach the domain from a number of directions, thus increasing the chances of revealing important material overlooked when other routes are taken. A third advantage to having access to several question types is their respective attention to different features. Each question is aimed at uncovering different aspects of the expertise. For example, questions from Cataloging the Categories focus on getting hold of the inventory of the expert's domain. Seeking Advice questions aim to disclose the expert's methods for hitting upon a solution.

The six types of questions also vary in their levels of specificity which, in combination, have the advantage of pulling out material varying in detail and composition. Grand Tour questions seek to draw out broad and comprehensive descriptions in order to grasp the scope of the problem domain, whereas Cross-Checking questions aim to verify fine detail. A fifth advantage to utilizing different types of questions is the ability to contain knowledge engineer bias. The range of questions should counteract the tendency of knowledge engineers to favor one type of question over another at the expense of a more comprehensive grasp of the relevant expertise.

Sixth, having access to a number of different types of questions guards against unwarranted assumptions by both knowledge engineer and expert. In asking a different type of question about the same material, the knowledge

26-10

engineer may find that he or she has incorrectly assumed certain connections or inclusions. Alternative questions may also encourage the expert to be more explicit about things previously assumed to be obvious. There are also a number of advantages having to do with doing well by the .expert. For example, having access to a number of different types of questions provides the knowledge engineer with opportunities to change direction or pace or content, and thus sustain the expert's interest and motivation. Multiple questions also indirectly convey to the expert that the knowledge engineer respects the range and complexity of what he or she knows. Finally multiple questions provides cross validation without CIOSS examination. By being able to draw on a range of queries, knowledge engineers can come back repeatedly to the same content area while avoiding the appearance of redundancy or skepticism.

#### STRATEGY FOR USING THE KNOWLEDGE ACQUISITION GRID

The five <u>Forms of Knowledge</u> and the six <u>Types of Questions</u> in the Grid do not exhaust either dimension. Rather each dimension is intended to provide the knowledge engineer with a wide-angled lens to see more of the relevant expertise. Moreover, the categories within each dimension overlap to some degree. For example, a Story about a particular situation may contain within it a Metaphor or Rule-of-Thumb. The important feature of the matrix nature of the Grid is not category independence but domain scope.

The Grid can be used both passively, in providing assistance in decoding expert's replies, and actively in deliberating evoking particular knowledge content. Each of these will be taken up in turn followed by some suggestions for structuring alternative paths through the Grid.

#### Decoding Experts' Replies

The Grid provides an explicit depiction of the reciprocal relationships that can exist between knowledge acquisition queries and experts' descriptions. Questions do not only elicit content but can follow it as well. Moreover, the Grid provides a way of dealing with replies which do not seem to be tied to a question as phrased. From the standpoint of the knowledge engineer, questions directed to experts do not always elicit the intended information. Nevertheless, a practiced knowledge acquisitor can make sense of the answer provided. The Grid provides a tool for appreciating and identifying the different forms in which expertise comes.

One of the marks of the good knowledge acquisitor is the ability to decode answers of all types and specially those which appear not to be a reply to the immediately preceding question. More specifically, knowledge engineers who use the Grid develop the habit of asking themselves after <u>each</u> reply, "To which question is this an answer?" and "Of what knowledge form is this an example?"

#### Evoking Particular Knowledge Forms

The Grid provides the means for deliberately evoking a particular kind of response from the expert. At times the expert may be asked for a Metaphor to open up a new line of investigation on the problem domain. Or a Story about an atypical situation may be solicited in order to get a different slant on the domain after a protocol analysis has been done on a story of a typical case.

Particular knowledge forms are evoked as a way to call up what is typically tacit knowledge. Since experts are frequently hard pressed to describe what they know, direct "why" or "how" questions are as frequently unsuccessful at uncovering important information. But if they can be given a mental handle, such as a request for a particular kind of experience with the domain, then previously implicit material can be made explicit.

#### Alternative Paths Through the Grid

Each dimension of the Grid has its own internal order. In the Types of Questions dimension, open-ended queries are at the top, while increasingly more specific, directed questions are located at the bottom. The Forms of Knowledge Dimension is somewhat less linear, but the more declarative kinds of knowledge are represented at the left, and the more procedural kinds of know-how are on the right side of the grid.

One way then of using the Grid is to start at the upper left and move back and forth in a more or less systematic manner to the lower right. Such an orientation begins with a broad investigation of the factual aspects of the expertise, and concludes with a focussed attention on specific heuristics.

This tactic is not, however, the only way to use the Grid. It can also be used in a more cyclic fashion. The knowledge engineer moves from general queries to specific ones and back to the general, and from the declarative to the procedural and back to the declarative in the knowledge mode. How the Grid is used depends ultimately on the particular needs of different expert systems teams; there is no single best way to deploy it. The value of the Grid is its multi-modal depiction of expert knowledge and the multi-form investigative means to get a hold of it.

#### SUMMARY

Skill with the Grid comes with practice. In the knowledge acquisition training module, participants learn how to use it through exposure to videotape examples as well as group exercises and guided feedback. Moreover, participants learn to use the Grid in the context of learning about other aspects of the knowledge acquisition process, including strategic, technical, and organizational concerns.

Manual knowledge acquisition can be made systematic. It can also be learned. The Knowledge acquisition training module has been developed to demonstrate both these properties. Traditionally, manual knowledge acquisition has been seen as putting in lots of unproductive hours. Experience with the Grid in particular has shown that greater sophistication with manual interviewing techniques can lead to time well spent and to greater overall knowledge acquisition proficiency.

#### References

- Abelson, R.P. (1981). The psychological status of the script concept. <u>American Psychologist</u>, <u>36</u>, 715-729.
- Anderson, J.R. (1980). <u>Cognitive</u> <u>psychology</u> <u>and its</u> <u>implications</u>. San Francisco, CA: Freeman.
- Chi, M.T.H., Glaser, R., Rees, E. (1982). Expertise in problem solving. In R. Sternberg (Ed.), <u>Advances in</u> <u>the psychology of human intelligence</u> (Vol 1, pp.7-75). Hillsdale, NJ: Lawrence Erlbaum.
- Chiesi, H.L., Spilich, G.J., & Voss, J.F. (1979). Acquisition of domain-related information in relation to high and low domain knowledge. <u>Journal</u> of <u>Verbal Learning and Verbal Behavior</u>, <u>18</u>, 257-273.
- Clark, H.H. (1985). Language use and language users. In G. Lindzey & E. Aronson (Eds.), <u>The handbook of</u> <u>social psychology.</u> (3rd ed., Vol 2, pp.179-232). Reading, MA: Addison-Wesley.
- Feigenbaum, E.A. (1980). <u>Knowledge engineering: The</u> <u>applied side of artificial intelligence</u>. Stanford, CA: Stanford University, Heuristic Programming Project.
- Larkin, J., McDermott, J., Simon, D.P., & Simon, H. (1980). Expert and novice performance in solving physics problems. <u>Science</u>, <u>208</u>, 1335-1342.
- Loftus, E.F. (1975). Leading questions and the eyewitness report. <u>Cognitive</u> <u>Psychology</u>, <u>7</u>, 560-572.
- Loftus, E. F. (1979). <u>Evéwitness testimony</u>. Cambridge, MA: Harvard University Press.
- Milojkovic, J.D. (1982). Chess imagery in novice and master. <u>Journal of Mental Imagery</u>, <u>6</u>, 125-144.

Minsky, M. (1986, in press). <u>Society of mind</u>

- Tulving, E. (1974). Cue-dependent forgetting. <u>American</u> <u>Scientist</u>, <u>62</u>, 74-82.
- Waterman, D.A., & Newell, A. (1971). Protocol analysis as a task for artificial intelligence. <u>Artificial</u> <u>Intelligence</u>, <u>2</u>, 285-318.
- Woodbury, H. (1984). The strategic use of questions in court. <u>Semiotica</u>, <u>48</u>, 197-228.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# ON COMPETENCE AND PERFORMANCE NOTIONS

# IN EXPERT SYSTEM DESIGN:

A Critique of Rapid Prototyping

Otto E. Laske

Arthur D. Little, Inc. Artificial Intelligence Center Acorn Park, Cambridge, Massachusetts 02140 U.S.A. TITLE ON COMPETENCE AND PERFORMANCE NOTIONS IN EXPERT SYSTEM DESIGN: A Critique of Rapid Prototyping.

AUTHOR: Dr. Otto E. Laske A.I. Center Arthur D. Little, Inc. 25 Acorn Park Cambridge, Massachusetts 02140 U.S.A.

ABSTRACT:

A distinction is made between four phases of knowledge engineering: the elicitation, analysis, encoding of knowledge, and prototype design. It is shown that present wisdom of how to work through these phases is defective in that it only yields two out of four possible types of expert system, and that to accede to the two remaining types of system, a different approach to knowledge engineering, especially knowledge elicitation, is required. This alternative approach entails decompiling large knowledge data bases (KDB's) produced via protocol analysis. The need for deferring commitment to architecture and inference mode is justified on cognitive-science grounds. It is shown that decompilation of KDB's enforces automation of knowledge analysis and encoding, fruitfully constrains knowledge elicitation, and undoes the present dichotomy of competence and performance notions in expert systems.

KEYWORDS: Knowledge elicitation, knowledge analysis, knowledge encoding, knowledge data base (KDB), KDB decompilation, automation of knowledge analysis and encoding.

# Modelling Human Expertise in Knowledge Engineering: Some P\_eliminary Observations

David C. Littman

Cognition and Programming Project Department of Computer Science Yale University New Haven, CT 06520 February 1986

# ABSTRACT

This paper reports the results of an empirical analysis of the knowledge engineering behavior of 6 persons with extensive experience in AI. The 6 persons were given the task of designing an AI program and were videotaped while they did so; during their two-to-three hour design sessions, they were asked to talk aloud about what they were doing and why they were doing it. The paper identifies several recurrent behaviors common to all the AI designers. For example, several components of the designers' goal structures are identified, as is the importance of focussing on a "touchstone", or key issue, around which much of the designer's behavior revolves. Several potential implications of the research for the design of knowledge engineering tools are explored.

# 1 Introduction: Motivation, and Goals

Knowledge transfer from expert humans to machines is slow, difficult, and prone to error. In order to automate the knowledge transfer process, knowledge engineers will have to write intelligent knowledge acquisition programs that are easily as complex as the domain programs they want to produce. One potentially useful approach to acquiring information that might assist developers of automated knowledge acquisition tools may be to

- study human expertise in knowledge acquisition as a skill in its own right and
- attempt to write programs that can reproduce it.

That is, it may be productive to view the problem of automated knowledge engineering as a knowledge engineering problem itself! Our goal, as "meta knowledge-engineers", would then be to identify the knowledge and skills that human knowledge engineers use when they perform the task of identifying an expert's underlying model of a domain. We would then engineer our knowledge about knowledge engineering into knowledge acquisition programs. It seems plausible to suppose that, if we could understand the skills and knowledge that knowledge engineers use when they build models of someone's reasoning in a domain (perhaps their own!), we could contribute to the goal of understanding how to build general automated knowledge acquisition tools. Ultimately, we would like to be able to write AI programs that can, themselves, write general knowledge engineering programs.

In this paper, we present preliminary analyses of videotaped protocols of 6 AI researchers who were given a transcript of a fragment of tutorial interaction between a tutor and a student in which the tutor was teaching the student some basic principles of large weather systems. The AI researchers were given the following task:

• Write "interesting" AI computer programs that simulate the interactions of the tutor and the student.

We gave our AI designers the task of writing simulation programs for two main reasons. First, building simulation programs is often a useful initial step in producing performance oriented knowledge based systems. Second, by giving our AI designers the goal of producing simulation programs, we forced them to focus their attention on the actual domain knowledge and rules of interaction that the tutor and student used. Forcing our AI designers to focus on the actual domain knowledge and tutorial knowledge appeared to have the effect of preventing them from looking for some AI "hacks" that would reproduce the tutorial interaction presented in the transcript of the tutorial dialogue without formulating a plausible representation of the student's and the tutor's knowledge.

Our goal in analyzing the protocols of the AI researchers as they designed their programs to simulate the interaction of the tutor and the student is guided by this hope:

• We hope that what we learn through systematic empirical study and simulation of human knowledge engineers will assist us in the knowledge transfer endeavor. We

2

further hope that, by understanding the knowledge and skills used by AI researchers to build AI programs, we will be able to build AI programs ourselves that will behave as AI researchers do when they are writing AI programs.

We realize that the observations presented in this note are preliminary. In addition, it may seem that much of what we observed in our subjects has been noted in books and papers such as Hayes-Roth, Waterman, and Lenat (1983) and Chandrasakaran (1985), or is part of the tacit knowledge of members of the AI community; both are true. The intended contribution of this work, however, is threefold:

- To suggest that knowledge engineering may itself be a reasonable domain for knowledge engineering
- To begin to provide an *empirical basis* for constructing intelligent knowledge engineering programs
- To suggest that the videotape protocol methodology may be a useful technique for assisting us in systematizing our empirical knowledge of knowledge engineering.

The preliminary conclusions from this work read like a bad-news good-news story. The bad news is that the amount of knowledge used by human knowledge engineers is enormous: building programs to simulate human knowledge engineers will be a formidable task. The good news comes in three parts:

- the program design behavior of different human knowledge engineers appears to be similar
- the design behavior of our AI designers appears to be consistent with models of the design process developed by researchers studying the cognitive underpinnings of software design<sup>1</sup>
- protocol methodology appears to provide useful insights into program design processes of knowledge engineers.

This note is thus intended to illustrate the potential value of the videotape protocol methodology for identifying some of the consistent, empirical aspects of the behavior of *human* knowledge engineers that may facilitate our work as we confront the task of building intelligent *machine* knowledge engineers. In Section 2 we describe the stimulus materials and methods we used in this experiment. In Section 3, we describe seven of the main themes we discovered in the behavior of our AI designers and present empirical data from the protocols of their problemsolving interviews. Finally, in Section 4, we present some implications, limitations, and future directions for our work.

# 2 Methods

#### 2.1 Subjects

The 6 subjects were experienced designers of AI programs. Each AI designer was an advanced student in the Yale Artificial Intelligence program. Each subject had at least six years of programming experience and had written at least one AI system containing more than 5000 lines of code. Most subjects had written several such AI systems and had worked on significantly larger AI systems.

# 2.2 Stimulus Materials and Procedure

Each subject was given a transcript of a fragment of dialogue between a tutor and a student and asked to design an AI program that would simulate the behavior of the tutor and the student. During the fragment of dialogue, the tutor teaches the student about certain aspects of large weather systems. The fragment of dialogue, which is included as Section 6, contains ten interchanges between tutor and student.<sup>2</sup> We chose this fragment of tutorial dialogue because the participants have quite different goals, as well as very different knowledge. The task appeared to engage the interest of each subject; each stated that the task was challenging, though they felt a longer transcript would have been useful. Each subject was videotaped while solving the AI design problem; the interviewer encouraged subjects to talk aloud during their interviews.

# 3 Central Themes in AI Designers' Performance

In this section, we present brief descriptions and examples of each of the following seven themes that appeared to be central to our AI designers' behavior:

- The Importance of the Knowledge Engineer's Goal Structure
- The Importance of World Knowledge
- The Selection of a General Representational Schema
- Causal Simulation of the Domain
- The Identification of Heuristics
- Model Testing Strategies and Progressive Refinement
- Focusing on a "Touchstone"

Each of the 6 subjects in our study exhibited several of these themes; we now briefly consider each theme in turn.

<sup>&</sup>lt;sup>2</sup>This transcript is a slightly modified version of a transcript presented by Stevens, Collins and Goldin in their paper "Misconceptions in students' understanding" in the book *Intelligent Tutoring Systems*, edited by D. Sleeman and J. S. Brown.

#### 3.1 Importance of the Knowledge Engineer's Goals

Each of our AI designers appeared to have a goal structure that guided the process of designing the simulation program. Five of the most important goals of the AI designers are:

- Identify the Domain Knowledge and Tutorial Knowledge: Subject 2 was most explicit about his need to have clearly in mind the knowledge that his program would have to represent. He was the most cautious of the designers: rather than jumping right in and selecting a representation for the knowledge, he spent most of his time just understanding the knowledge his program would have to manipulate. Subject 1, Subject 2, Subject 3 and Subject 4 all spent considerable effort understanding the tutorial knowledge of the tutor so that they could understand how the tutor constructed responses to the student.
- Use a Consistent Representation: Subject 1 was most explicit about his enforcement of the constraint of using a consistent representation for the knowledge in his program. Subject 1 realized that a common representation for the knowledge of the tutor and the student would facilitate the communication between the two parties in his simulation of the tutorial interaction. Interestingly, the heuristic of using a common representation led Subject 1 to ignore the possible desirablity of using different representations for the tutor and the student; thus, he did not question the psychological reality of using the same basic representation.
- Identify and Solve Key Problems: All of our subjects identified key issues that their programs would have to address. Some key issues were problems of representing knowledge; other key issues related to how to organize the control structure of the program.
- Use Stepwise Refinement: Each of the subjects recognized the need to develop their designs so that the degree of specificity of each of the components of the program stayed "in synch". Subject 3 was most articulate about this issue when he noted that he would design a modular program and said that he would "go back and forth" between the modules as he fleshed out the design. The strategy of going back and forth apparently served to prevent designers from making commitments to one aspect of the design that would make other aspects difficult to design.
- Identify the Abstract Structure of the Problem: Subject 3 explicitly used the method of abstraction to help him design his program. As we illustrate in Section 3.5, Subject 3 abstracted away from the domain the pattern of the tutorial interaction.

These five components of the goal structures of our AI designers were important aspects of their design processes. A tentative implication for knowledge engineering design tools of the importance of the designer's goal structure may be that such tools should provide the designer with a mechanism for directly expressing and fulfilling his or her goals.

# 3.2 Importance of World Knowledge

The importance of world knowledge for the process of constructing intelligent knowledge-based systems cannot be overemphasized. We repeatedly observed our subjects appealing to their world knowledge as they designed their programs. Our subjects appealed to world knowledge in two main ways.

Appeal to Domain Knowledge: In order to understand what the student understood, and how the tutor generated responses to the student, all subjects referred extensively to their knowledge of the weather domain. In addition, almost every subject regretted not knowing more than they did know about the weather domain. For example, Subject 1, who had extensive training in physics, noted several times the limitations of his knowledge about weather systems. On one occasion he simply did not know the reason for the form of the tutor's response to the student. Subject 1's puzzlement over the goal the tutor was trying to achieve with the reply to the student occasioned this comment:

"Obviously my model of weather is off. It's a confounding factor. The first thing I'd do in building (this) AI system would be to get out some books on weather and understand what is really going on."

This quotation by Subject 1 illustrates the importance to knowledge engineers, and to programs which would simulate their behavior, of a reasonably powerful model of the domain in which they are engineering knowledge.

Appeal to General World Knowledge: Our subjects frequently appealed to world knowledge that was not related directly to the weather domain. For example, one of our subjects noted that transcripts of dialogue sometimes failed to capture potentially useful information, such as the motivation and mood of the participants. Another subject pointed out that different tutors might have very different styles for interacting with students; equally, some subjects observed that motivational differences of students might require their tutorial programs to reason about differences in students' learning style. Such differences in tutoring and learning styles have an impact on the design of the AI program since they would either have to be simulated or ignored altogether.

# **3.3 Selection of General Representational Schema**

Early in the process of constructing their AI programs, both Subject 1 and Subject 3 settled on general frameworks for conceptualizing the rules and knowledge of the weather domain. For example, Subject 1 decided, very early on in his analysis of the transcript of the weather tutoring session, that he would use a frame-slot notation for the weather domain knowledge of *both* the student and the tutor. Approximately two minutes after beginning his task, Subject 1 identified his representation:

"If there's a climate frame, one of its slots is going to have to be temperature."

Subject 1's statement is consistent with his subsequent analysis of the requirements of his program's representation of the weather domain. He continued to pursue the applicability of the frame-slot representation, identifying the kinds of knowledge the frames would have to represent as well as heuristics for filling the slots with values. Thus, Subject 1's early decision to use a frame-based representation for the weather domain knowledge played an important role in the

subsequent design work of his AI program.

Subject 3 initially settled on a causal-chain representation for the knowledge in the weather domain.<sup>3</sup> and this had a strong impact on his subsequent work. While Subject 3 recognized that his initial causal-chain representation of weather domain knowledge was not entirely adequate, he based most of the machinery for the tutor's question-generation behavior directly on the causal-chain representation. For example, the causal chain representation would permit the tutor to work back along causal chains to help students explain how observed effects arose (e.g., precipitation) and to work forward along causal chains to help students learn to predict what would happen given a particular state of the world (e.g., water-burdened clouds running up against a mountain.)

Thus, while Subject 1 and Subject 3 selected different initial representations for weather domain knowledge, their early selections of representations strongly affected their subsequent designs of their programs.

# 3.4 Causal Simulation of the Domain

In building their representations of the weather domain, our subjects used naive physical process theory to mentally simulate the causality in the weather domain as they tried to understand what the student understood and what the tutor was trying to teach. The following quotation illustrates Subject 1's use of mental simulation to understand the effect of the Japanese Current on the climate in the Northwest United States: Subject 1 performed this reasoning when he was trying to understand why the tutor asked the student how the Japanese Current affects rainfall in Washington and Oregon.

"I'm inferring that the Japanese Current is cold. If I make (this) assumption about the Current's path, then the Current starts out in Japan, which is fairly far north, and goes even farther north. So, it probably gets very cold when it comes down the coast and probably makes Washington and Oregon a fairly cold place."

Though Subject 1 made the incorrect assumption that the Japanese Current is cold, this quotation illustrates the use of simulation to reason about causality as part of the process of designing the behavior of the model of the student and the tutor. While some subjects performed simulation more than others, every subject used simulation of the domain during their program design behavior.<sup>4</sup>

<sup>4</sup>See Littman & Soloway (1986) for a more extensive treatment of mental simulation in design and understanding of programs.

<sup>&</sup>lt;sup>3</sup>A causal-chain representation relates causes in the weather domain (e.g., air masses cooling) to their effects (e.g., releasing moisture as precipitation).

# **3.5 Identification of Heuristics**

Subjects identified two kinds of heuristic reasoning that their programs would have to perform. First, subjects identified several heuristics for reasoning in the weather domain; second they identified several program control heuristics that the tutor and the student used to control the tutorial session.

Domain Heuristics: Subject 1 interpreted the student's behavior in terms of his frame representation for the student's knowledge about climate. Subject 1 proposed that the slots of the student's climate frames were filled by applying heuristics which could include default methods for filling slots.

"The second heuristic is that the slots have metrics associated with them with distributions on the values and a heuristic for selecting a value out of that distribution based on the "is-part-of" relations for geographic areas."

Heuristics for Controlling Program Behavior: Subject 1 was very interested in understanding how the tutor constructed responses to the student, since he viewed this as the primary aspect of his program's control mechanism. After reading and analyzing several interchanges between the student and the tutor, Subject 1 said:

The interviewer is definitely building this question (to the student) out of the student's responses.

Then, after seeing a few more interchanges, Subject 1 had a flash of insight and stated the tutor's heuristic for constructing replies to the student:

"If the student comes up with a false statement, the statement is immediately contradicted in an interesting way. It's (the tutor's response) not just "NO." ...(the response) relates back to a previous question of the interviewer; the correct response (to that question) is what the interviewer says now."

Thus, Subject 1 identified two kinds of heuristic reasoning for his program to perform, one related to the domain, the other related to the control structure of his program. It is interesting to note that both domain and program control heuristics were based on the frame-slot representation Subject 1 had selected: Subject 1 designed the domain heuristics to *fill* slots in the frame; he designed the program control heuristics to use information in the frames.

Subject 3 also focused on the heuristics used by the tutor and student to construct their responses. Subject 3 abstracted the transcript of the tutorial interaction by categorizing each of the tutor's and each of the student's statements; the designer's categorization of the statements in the transcript produced a foundation for a schematic description of the types of utterances the tutor and student made. Using the schematic description of the student's and tutor's utterances, Subject 3 then attempted to identify heuristics the tutor and the student used to produce a response of each type.

# 3.6 Model Testing and Progressive Refinement

All of our subjects tested their evolving models of the tutor and student at various stages in the design process. On the basis of their tests of their evolving models, they refined and corrected their program designs.

Model Testing: All subjects noted the importance of testing their evolving models of their programs. Most subjects used a variant of the "predict-and-verify" method. Here is how Subject 1 expressed his strategy:

"So I guess I am playing this game, predicting what the interviewer is trying to say."

The important observation to make about the skills of progressive refinement and testing the adequacy of the model is that both require a great deal of heuristic knowledge. For example, how did our designers know:

- which components of the design should be tested?
- when to test components of the design?
- how to test different components of the design?
- how to interpret the results of the tests?<sup>5</sup>

The last point, interpreting the results of the test of the model, points out the importance to our subjects of debugging their designs. For example, Subject 1 spent a great deal of time resolving contradictions between his representation of the tutor's knowledge of the Japanese Current and his factual knowledge about the fate of moisture in bodies of air that are cooling. The following quotation illustrates Subject 1's concern with identifying the sources of contradictions. He said this:

"I get a contradiction because I know that if things cool off they lose their moisture. So I know there is a lot of rainfall in Washington and Oregon. But I am thinking that if the (Japanese) Current is coming down from the North it will already have lost its precipitation so I get a contradiction."

In fact, Subject 1 had made a faulty assumption about the Japanese Current (it is actually a warm body of water) and built the faulty assumption into his representation of the tutor's knowledge. Subject 1's faulty representation of the tutor's knowledge of the Japanese Current led him to a contradiction between what he believed about the Japanese Current (that it is cold) and the affirmative response of the tutor to the student's assertion that it is probably rainy in the Pacific Northwest because of the effect of the Japanese Current on the precipitation patterns. Subject 1 attempted to resolve this contradiction several times, putting it aside each time until some new piece of evidence became relevant to its resolution. The behavior of "putting aside" the contradiction, until some new information became relevant to its resolution, is consistent with the model of software design proposed by Adelson and Soloway (1985) in which "demons" help

<sup>5</sup>The credit assignment monster raises its head here.

expert software designers turn their attention to old issues when new information becomes relevant to them.

**Progressive Refinement:** Our subjects began their AI tasks with only a very general idea of how their programs should behave. After an initial phase of identifying important attributes of the student's and the tutor's knowledge and reasoning, they decided on a potential representation for the knowledge and reasoning. As our subjects filled in their designs, they refined their initial representations by a three-step process:

- derive predictions about the behavior of the tutor and the student from their program designs
- evaluate the predictions with the transcript of the tutorial session
- alter the model to be consistent with the transcript.

Our subjects knew that their programs would have to be designed in a "propose-test-modify" cycle. Subject 3, for example, was fully aware that his first attempt to design the program would be primarily a "throwaway". Subject 3's recognition that much of his initial program would be a throwaway is apparent in this comment:

"It's always a problem: How far to expand something you know you are going to throw away."

#### 3.7 Focusing on a "Touchstone"

After studying each of the statements of the tutor and the student, our AI designers had a general idea of the knowledge the AI program would have to include and the rules it would have to use to manipulate the knowledge effectively. Following this initial assessment of the requirements of the program, several of our AI designers identified a tutorial interaction, or a response, that they considered to be crucial for their programs. Subject 2, for example, decided that the third interaction, in which the tutor asks the student how the Japanese Current affects rainfall in the Pacific Northwest, was crucial for his program. He said that if he could understand how to generate the tutor's question, and how to generate the student's response, then he would understand:

- how to represent the domain knowledge
- how to represent the tutor's strategies for asking questions
- how to represent the student's strategies for building informative responses.

A touchstone is valuable because it provides a test for the essence of something. The skill of selecting touchstones for testing knowledge-based programs is important; it permits the designer of the program to identify what would count as an effective solution to the problem. If we understood better how knowledge engineers know "what counts as a good solution", we may be able to design support tools to assist them in developing such solutions.

# 4 Implications, Limitations, and Future Directions

## 4.1 Implications

The current work suggests four main implications.

- The means by which expert knowledge engineers extract mental models of domain experts is, itself, heavily knowledge based.
- A great deal of the behavior of experienced knowledge engineers appears to be based on *heuristic classification* of problem types (Clancey, 1985). For example, the AI designers in our study who had some experience studying or designing tutoring systems made a commitment to a general representation schema earlier in their design processes than the AI designers who had never worked in the domain of tutoring systems. The experience of the AI designers who had worked in the domain of tutoring systems appeared to permit them to classify the problem type quite readily and to select a representational system based on the problem type. Thus, it would be very useful to study the organization of our knowledge engineers' knowledge.
- The process of empirically studying the methods that knowledge engineers use to extract mental models of domain experts is a potentially useful enterprise.
- The videotape protocol methodology appears to provide a potentially useful tool for studying the process of designing knowledge engineered programs.

Each of these implications must be explored by future research before it is possible to evaluate them fully.

# 4.2 Limitations

The information presented in this paper does not constitute even an attempt to formulate an initial empirical theory of knowledge engineering. Rather, it attempts to show that describing consistent aspects of the behavior of knowledge engineers can suggest interesting avenues to pursue toward the goal of developing computational theories of how knowledge engineers do their work. The ideas presented in this note are limited for three main reasons:

- the subject sample is small
- the domain in which the AI designers were asked to work is limited: a more adequate empirical descriptive theory of knowledge engineering should study behavior of knowledge engineers in several domains
- the results reported here are just a first pass at analyzing the extensive protocol data collected from our subjects' design sessions.

Even with these limitations firmly in mind, it seems plausible to imagine that a more detailed empirical study of the behavior of knowledge engineers could provide useful insights into ways of devising intelligent, knowledge-based knowledge engineering tools.

# 4.3 Future Directions

In the future, we plan to address several issues:

- More Detailed Analysis of the Data: The protocol data we collected are very rich. The goal in performing further analyses will be to identify
  - the actual knowledge used by our subjects
  - ▶ the goal structure of our AI designers that controlled their design behavior
  - ▶ the heuristics they used to achieve their goals.
- Develop a Process Model: In this paper we have identified just a few of the salient behaviors of our designers. As we identify more of the knowledge and heuristics used by our designers, we plan to develop a process model of their behavior.
- Develop a Simulation: As the process model of our designers' behavior becomes fleshed out, we will develop a simulation model of their behavior.

#### 4.4 Acknowledgements

I would like to thank Elliot Soloway for his unflagging support of my work. James Spohrer and Andy Liles made very useful comments on drafts of this paper. I would also like to extend my appreciation to my subjects.

#### 5 References

Adelson, B. and Soloway, E. A model of software design. Yale Computer Science Department Technical Report #342, 1984, Department of Computer Science, Yale University, New Haven, CT.

Chandrasakaran, B. Generic tasks in expert system design and their role in explanation of problem solving. Technical Report, July, 1985, Department of Computer and Information Science, Laboratory for Artificial Intelligence, The Ohio State University, Columbus, OH.

Clancey, W. Heuristic Classification. Stanford University Computer Science Department Research Report #STAN-CS-85-1066.

Building expert systems. F. Hayes-Roth, D. Waterman and D. Lenat (eds.), Addison-Wesley, Reading, MA., 1983.

Littman, D. & Soloway, E. Mental simulation of programs: Some data and some theory. Submitted to 1987 Conference on Computer Human Interaction (CHI).

Stevens, A., Collins, A., and Goldin, S. Misconceptions in students' understanding in Intelligent tutoring systems D. Sleeman and J. S. Brown (eds.), Academic Press, London, 1982.

# 6 Stimulus Transcript

The following transcript is the actual transcript shown to our subjects.

tutor: 1. Do you know what the climate is like in Washington and Oregon?

student: 1. Is it cold?

tutor: 2. Can you guess about the rainfall there!

student: 2. Normal, I guess.

tutor: 3. How do you think the Japan Current along the coast affects rainfall there?

student: 3. It's probably rainy.

tutor: 4. Yes. There are rain forests in Washington.

student: 4. Then the Japanese Current is warm.

tutor: 5. Right.

student: 5. And the wind blows in from the sea.

tutor: 6. What happens to the moist air blown over Washington and Oregon by the winds?

student: 6. It condenses and rains because the moist air cools and cannot hold the water. It cools when the wind blows it and it lowers from the sky.

tutor: 7. What happens to the temperature of the moist air as it rises?

student: 7. The moist air gets warm.

tutor: 8. No, it cools when it rises.

student: 8. So the mountains also cool the moist air when they come into contact with the moist air.

tutor: 9. No. The contact with a solid mass does not cool off moist air.

student: 9. Because it is not big enough.

tutor: 10. Right.

student: 10. So then it is because the mountains cause the air mass to rise?

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# Taking Backtracking With a Grain of SALT

Sandra Marcus Department of Computer Science Carnegie-Mellon University Pittsburgh, Pennsylvania 15213\*

In addition to the author, Emile Servan-Schreiber contributed significantly to the coding of the SALT analyses described in this paper. Many of the ideas contained here were presented in a discussion group attended by Larry Eshelman, Gary Kahn, John McDermott, Tom Mitchell, and Allen Newell, and I am very grateful for their feedback. I would also like to thank Jeff Stout for his work on the problem-solving strategy and control shell assumed by SALT.

# Introduction

A very successful approach to automating knowledge acquisition has been to focus a tool on expert systems that use the same problem-solving method (Davis and Lenat, 1982; Boose, 1984; Marcus, McDermott, and Wang, 1985; Kahn, Nowlan, and McDermott, 1984; van de Brug, Bachant and McDermott, 1986). Such tools gain their advantage from the way in which they represent the knowledge they acquire; that is, the representation can identify knowledge by the role it plays in finding a solution (Clancey, 1985; McDermott, 1986). Knowing how knowledge will be applied provides a strong lever for managing the input of the knowledge and deciding the appropriateness of the knowledge base for the intended use. This paper will describe some aspects of user guidance provided by the knowledge acquisition tool SALT (Marcus, McDermott, and Wang, 1985; Marcus and McDermott, 1986), based on a knowledge representation for constructive applications for which a propose-and-revise strategy is appropriate.

SALT assumes that any expert system it creates will use a strategy of incrementally extending a plan while monitoring constraints on plan extensions. Whenever it detects a constraint violation, the system will use domain expertise to consider past decisions that could be changed, choose the least damaging change that remedies the violation, remove anything potentially inconsistent with that change, and continue extending the plan from that point. For an unaided domain expert to describe how to perform a task using backtracking involves making explicit a great deal of control knowledge, which experts are usually fairly vague about. It is particularly difficult for the expert to keep in mind how steps to extend the plan should be ordered, which decisions a fect compliance with constraints on the plan, and how local refinements that remedy individual constraint violations interact with each other to affect overall progress toward a plan that meets *all* constraints. The broad outlines of the method SALT assumes define a knowledge representation SALT can exploit to assist users with these difficulties. SALT's knowledge base analyses can indicate when backtracking is needed, identify potential points to backtrack to, and indicate potential problems in converging on a solution that may require additional knowledge.

Dr. Marcus is now with Boeing Computer Services, Advanced Technology Center, P.O. Box 24346, M/S 7L-64, Seattle, WA 98124.

# Is Backtracking Needed?

Experts in design and planning are fairly good at describing individual considerations for constructing a solution for their domain. They can extemporaneously list many of the constraints the solution must satisfy. They can consult manuals of formulas and tables for producing individual output values. But they are less clear on how the individual steps fit together, how to organize them into a system. SALT aids users by allowing them to enter knowledge piecemeal starting at any point. SALT keeps track of how the pieces are fitting together and warns the user of places where pieces might be missing or creating inconsistencies.

SALT users typically start by specifying procedures for producing particular desired output values. They can start by specifying a procedure for any value. Users can describe a set of procedures from input to output if they wish to do so. Alternatively, users may start by giving a procedure for determining an output value that takes into consideration all information necessary to make a decision. SALT can then march the user backward to acquire procedures for the information necessary at that step, and so on. If each step includes all relevant considerations, the resulting expert system will be a least commitment system in the sense that, like MOLGEN (Stefik, 1981a; 1981b), it will not make a decision until all necessary information is available. If steps cannot be ordered so as to make all information available before a decision is required, SALT will guide the expert in combining plausible reasoning to propose an initial guess with knowledge-based backtracking to revise guesses. Examples based on VT (Marcus, Stout, and McDermott, 1986), an expert system for configuring elevators, can illustrate how SALT exploits its knowledge of proposeand-revise problem-solving to guide the expert. These examples will follow a user whose strategy is to start with a procedure for specifying one of the desired output values of the expert system.

VT takes as input the customer's functional specifications, such as elevator speed and carrying capacity, and key architectural information about the elevator shaft, such as wall-to-wall dimensions and locations of sills and rail supports. VT's task is to select all pieces of equipment and produce a configuration and layout of parts that meet both those specifications and safety, installation and maintenance requirements.

At the start of a SALT interview, the user is shown the menu below for indicating the type of knowledge to be entered or viewed. Three basic kinds (or roles) of knowledge make up a propose-and-revise system: (1) procedures for proposing values for the pieces of the design or plan the system will output, (2) identification of constraints on individual pieces of the design or plan, and (3) suggestions for ways of revising the design if the constraints are not met.

In the interaction below, the user is electing to enter a procedure for determining the value of MACHINE-MODEL. SALT's messages are in sans serif type, user replies in sans serif italic. A word in brackets at the end of a SALT request for information is a default response which the user can issue by typing carriage return. (These conventions apply to all sample SALT dialogs in this paper.)

1.	PROCEDURE	Enter a procedure for a value
2.	CONSTRAINT	Enter constraints for a value
2	EIV	Enter remarking for a constraint violat

- 3. FIX Enter remedies for a constraint violation
- 4. EXIT Exit

Enter your command [EXIT]: 1

What is the name of the value for which you will specify a procedure?: MACHINE-MODEL

SALT next presents the user with a set of prompts. These are requests for information needed to specify a procedure for the SALT-assumed problem-solving strategy. The completed PROCEDURE schema for MACHINE-MODEL is as follows:

1	Name:	MACHINE-MODEL
2	Constraint type:	ACTUAL
3	Precondition:	NONE
4	Procedure:	DATABASE-LOOKUP
5	Table name:	MACHINE
6	Column with needed value:	MODEL
7	Parameter test:	MAX-LOAD > = SUSPENDED-LOAD
8	Parameter test:	NONE
9	Ordering column:	HEIGHT
10	Optimal:	SMALLEST

Enter your command [EXIT]: <cr>

The user's choice of DATABASE-LOOKUP as the procedure type for determining MACHINE-MODEL indicates that the SALT-generated expert system will consult its database of equipment specifications to select an appropriate machine.<sup>1</sup> With this procedure, the user is presented with a set of subprompts asking for details for locating the value to be retrieved. In the schema above, the name of the table and column from which the value is retrieved are SALT-generated defaults. Parameter tests are tests to be performed on table entries (rows) to decide which are appropriate candidates for retrieval. In this case, the entry must have a listing under the column MAX-LOAD that is greater than or equal to the SUSPENDED-LOAD. Finally, if more than one entry under MODEL meets this test, ordering column and optimal considerations are used to determine which entry should be retrieved. The user indicates that the entry with the smallest height is the most desirable.

This procedure in isolation is incomplete. SALT notices this and makes the query below if the user tries to exit from the interview at this point:

I have no PROCEDURE for arriving at a value for SUSPENDED-LOAD which was mentioned as a contributor for MACHINE-MODEL. Do you wish to specify one now? [SAVE]:

The following three procedures are supplied by the user in response to SALT prompts to supply missing knowledge.<sup>2</sup>

1	Name:	SUSPENDED-LOAD
2	Constraint type:	ACTUAL
3	Precondition:	NONE
4	Procedure:	CALCULATION
5	Formula:	CAR-WEIGHT + CABLE-WEIGHT

Enter your command [EXIT]:

<sup>1</sup>SALT can also be used to collect and generate a database that is organized into tables of constants. The user will be reminded to enter constants for any table referenced in a procedure.

<sup>2</sup>This is a simplification of an example that occurred as SALT acquired the VT knowledge base. The original analysis involved a chain of 13 steps.

- 1 Name: CABLE-WEIGHT
- 2 Constraint type:
- 3 Precondition:
- 4 Procedure:
- 5 Formula:

ACTUAL NONE CALCULATION HOIST-CABLE-UNIT-WEIGHT \* HOIST-CABLE-QUANTITY

Enter your command [EXIT]:

1	Name:	HOIST-CABLE-QUANTITY
2	Constraint type:	ACTUAL
3	Precondition:	NONE
4	Procedure:	CALCULATION
5	Formula:	SUSPENDED-LOAD / HOIST-CABLE-STRENGTH

Enter your command [EXIT]:

Once the user enters these procedures, SALT stores that knowledge within a dependency network shown below:



Using this representation, it is very easy for SALT to detect the impossibility of applying three of the steps in a forward chain. SALT's aid in this situation is guidance in how to use propose-and-revise instead. First the user is warned that the situation exists:
In the procedures I have been given, there is a loop. The list below shows the values on the loop; each value uses the one below it and the last uses the first:

- 1 CABLE-WEIGHT
- 2 HOIST-CABLE-QUANTITY
- 3 SUSPENDED-LOAD

In order to use any procedure, I need some way of getting a first estimate for one of the names on the list. Which one do you wish to estimate? [SUSPENDED-LOAD]: 2

The user is asked to break the loop by providing an estimate for one of the values on the loop. The user indicates that an estimate will be provided for HOIST-CABLE-QUANTITY by typing its number, 2.

The format used for eliciting an estimate is the same as that used for any step to extend a plan:

- 1 Name: HOIST-CABLE-QUANTITY 2 Precondition: NONE
- 3 Procedure: CALCULATION
- 4 Formula:

Enter your command [EXIT]: 4 3

Prompts on the left represent requests for information. Values on the right in the example are defaults. As shown below, the user indicates that in extending any plan, the system should initially use three hoist cables. SALT users are coached to use the most preferred value as an estimate. In this case, the smallest number of hoist cables that can be used on any job incurs the smallest possible dollar cost.

1	Name:	HOIST-CABLE-QUANTITY
2	Precondition:	NONE
3	Procedure:	CALCULATION
4	Formula:	3

Enter your command [EXIT]: <cr>

Once the user has settled on this procedure and exited from the procedure screen, SALT indicates that this will be a potential point for initiating backtracking, since the original procedure will be used to derive a constraint placed on this estimate. In this case, the user accepts the default, MINIMUM:

The procedure you originally gave for HOIST-CABLE-QUANTITY will be used as a check of the estimate. How does the value arrived at by that procedure limit the estimate? [MINIMUM]: < cr>

SALT's representation of the revised knowledge base that results from this interaction is shown below:



The system is now set up to start with three hoist cables; use this value to compute CABLE-WEIGHT, then SUSPENDED-LOAD; and finally, compute MINIMUM-HOIST-CABLE-QUANTITY. If the computed minimum is greater than three, the system will register a constraint violation. SALT has essentially taken knowledge that the user originally defined as filling the role of extending a plan and assigned it as part of the system's constraint knowledge. Because SALT now has knowledge of a new constraint, it must have a way of remedying a violation of that constraint, and it reminds the user:

I have no knowledge of fixes for MINIMUM-HOIST-CABLE-QUANTITY. Would you like to specify one now [SAVE]? yes

The next section describes aid SALT can provide in suggesting potential fixes.

## Where Should the System Backtrack To?

In many cases, it can be difficult to decide what parts of the proposed plan to revise in order to remedy a constraint violation. In principle, any value that contributes to the constraint or its constrained value might serve as a potential fix value. This is the rationale behind dependency-directed backtracking, applied in expert systems such as EL (Stallman and Sussman, 1977; Sussman and Steele, 1980). But revisions may differ in cost and in likelihood of success. SALT assumes a method of *knowledge-based* backtracking; that is, the generated system will use domain expertise to decide

what values to change to remedy a constraint violation.<sup>3</sup> A user who wishes to suggest revisions in response to a constraint violation should at least consider any contributor that could have an impact. If the dependency network is very dense, the user may have difficulty recalling all contributors. SALT helps by reading out the relevant part of the network on request. The example below is for the knowledge base described so far, after the user elects to supply a fix for the MINIMUM-HOIST-CABLE-QUANTITY constraint.

There are no fixes for MINIMUM-HOIST-CABLE-QUANTITY in the knowledge base. Would you like to see possible fix values? [YES]:

There are no contributors to HOIST-CABLE-QUANTITY.

Would you like to see the contributors to MINIMUM-HOIST-CABLE-QUANTITY? [YES]:

Contributors to MINIMUM-HOIST-CABLE-QUANTITY:

- 1 HOIST-CABLE-STRENGTH
- 2 SUSPENDED-LOAD
- 3 CABLE-WEIGHT
- 4 HOIST-CABLE-QUANTITY
- 5 HOIST-CABLE-UNIT-WEIGHT
- 6 CAR-WEIGHT

Give the number of the one you want to work on (0 for new) [0]:

Because HOIST-CABLE-QUANTITY was assigned a constant value of 3, there are no contributors to it. The indentation in the display of contributors to MINIMUM-HOIST-CABLE-QUANTITY indicates the remoteness of their contribution. The leftmost contributors, SUSPENDED-LOAD and HOIST-CABLE-STRENGTH, are used directly to calculate the constraint. CABLE-WEIGHT and CAR-WEIGHT, indented under SUSPENDED-LOAD, directly contribute to SUSPENDED-LOAD and thus to MINIMUM-HOIST-CABLE-QUANTITY. In principle, a change in any contributor to MINIMUM-HOIST-CABLE-QUANTITY, however remote, could change its value and, depending on the direction of change, remedy the constraint violation. Domain expertise is needed to determine what change to try because the domain expert knows which changes are most disruptive or expensive in the real-world, and also which are most likely to succeed.

In this example, the user does not suggest changing any of the contributing values but would revise HOIST-CABLE-QUANTITY directly. The proposed fix is shown below:

1	Violated constraint:	MINIMUM-HOIST-CABLE-QUANTITY
2	Value to change:	HOIST-CABLE-QUANTITY
3	Change type:	INCREASE
4	Step type:	SAME
5	Rating of undesirability:	4
6	Reason for undesirability:	Changes minor equipment selection

SALT's representation of the knowledge base now looks as shown below:

<sup>3</sup>AIR-CYL (Brown, 1985) and PRIDE (Mittal and Araya, 1986) also use advice for recommending revisions but differs from SALT-generated systems in other architectural features.



The knowledge base now calls for the generated system to start with the smallest possible number of hoist cables and use that estimate to make other equipment selection and sizing decisions. The system will then use the results of those decisions to calculate the smallest number of hoist cables required under those conditions. If the minimum is three or fewer, the current configuration is acceptable. If it is greater, HOIST-CABLE-QUANTITY will be increased by the amount that it fell below the minimum, and the process will be repeated using the new estimate for HOIST-CABLE-QUANTITY. If the calculated minimum ever exceeds the specified maximum, the system will stop increasing HOIST-CABLE-QUANTITY and reach a dead end. It will then declare that no solution is possible for this over-constrained job.

In this example, the search carried on by the mechanism that implements fixes is extremely simple. Since there is only one potential remedy for MINIMUM-HOIST-CABLE-QUANTITY, the generated system need not decide which potential fix to apply. When there is more than one possible fix, the system must choose among them in a way that ensures that the system will converge on a solution if one exists. SALT provides a map of the knowledge base to help sophisticated users assess the adequacy of any knowledge base collected by SALT. This map and its use are described in the next section.

# Will the Search Converge?

A piece of knowledge supplied by the user to fix a violated constraint carries with it three crucial bits of information: a value that might be changed, the way it should be changed, and the cost of the change. The first two pieces of information give the system potential remedies for constraint violations. The third influences the order in which the system attempts to apply those remedies.

Domain knowledge determines the relative cost of potential constraint remedies. For VT, ordering largely reflects dollar cost to the company responsible for supplying, installing, and maintaining the equipment. Potential fixes are ordered according to their most severe negative effect on the configuration. For VT, potential fixes are investigated in that order, starting with the least costly, because it is very important to the elevator company not to overlook any low-cost solutions. However, this method excludes any use of information that might reflect the likelihood of convergence on a solution that satisfies *all* constraints. Evaluating the likelihood of convergence requires examining the effect of a particular fix on other potential constraint violations.

The kind of domain-specific information SALT collects to direct backtracking is relatively easy to elicit, since the expert can focus on one constraint violation at a time and give preference ratings that reflect cost considerations alone. However, a search that relies solely on this local information and ignores potential interactions among fixes for different constraint violations may run into trouble. SALT helps manage knowledge-based backtracking by mapping out interactions among fixes for different constraint violations. Interacting fixes can then be examined to see if they might interfere with convergence on a solution. Fixes that do not cause problems can be dealt with using local information only. This treatment, the "standard treatment" for SALT-generated systems, ignores potential interactions among fixes for different constraints. Trouble spots are treated as special cases that take into account more global information.

## The Assumed Standard Treatment and Its Trouble Spots

In the standard treatment, choice of where to backtrack to is conditioned on individual constraint violations. Only potential fixes identified by the domain expert as relevant to the current violation are considered, and these are selected in order of the expert's preference. All possible combinations of constraint-specific remedies will be tried until a remedy is found for the violation. If the system reaches a dead end -- i.e., none of these combinations remedy the local constraint violation -- the system will announce that there is no possible solution. If fixes for one constraint violation do not affect other constraint violations, this strategy guarantees that the first solution found will be the most preferred and that the system will correctly report failure if no successful fix is found for an individual constraint.

However, it is possible that remedies selected for one constraint violation may aggravate constraint violations that occur further downstream. This may result in failure to find a solution when one does exist in cases in which a fix that appears optimal based on local information would not be preferred if more were known about the search space.

For example, the lowest cost successful fix for a particular constraint violation may aggravate a downstream constraint violation so much that the system reaches a dead end when exploring fixes for these downstream violations. If more expensive fixes for the first constraint do not have the same negative effect downstream, nudging the search through the more expensive first step may give a solution. The undesirable behavior of the system in this case would be a premature announcement of failure.

Another potential problem is that unproductive looping may occur between fixes for two constraint violations if their preferred fixes counteract each other.<sup>4</sup> This will occur, for example, if fixing one constraint violation increases a certain value, which leads to violation of another constraint whose fix calls for decreasing the same value, and so on. Repeated violations of the same constraint are not necessarily pernicious, but such a case of antagonistic constraints might result in an infinite loop.

Currently SALT displays a representation of the interactions among fixes in a knowledge base. This was used this to analyze VT's knowledge base for potential trouble in a local, constraint-specific search. Special-case treatments for problem spots were then hand-coded. Plans for automating this process in SALT are discussed under "Future Directions."

## **SALT's Map of Fix Interactions**

In order to identify trouble spots, we need to know whether and how a proposed change to a fix value will affect the relationship of constraints and their constrained values anywhere else in the network. To map these interactions, SALT uses a worst case analysis. The dependency network that represents contribution in procedures for proposing plan extensions and specifying constraints can be augmented in many cases by a notation for direction of correlation between values at adjacent nodes. For many algebraic formulas, SALT can assess whether an increase in a contributor will produce an increase in the result or the reverse. For example, if z = x - y, increasing x will increase z, while increasing y will decrease z. Procedures involving calls to the database cannot be similarly assessed because, by design, what is stored in the database could vary on a very short-term basis. Thus, anything involving a database call is marked unknown and considered to have a potentially aggravating effect on downstream constraint checks.

Most fixes have an associated direction -- increase or decrease of the fix value. Using the correlations, this direction of change can be propagated through the network to any constraints or constrained values. As constraints involve relational tests between values, knowing the propagated effect on the constraint or constrained value will tell us whether the change to some upstream fix value will make the constraint more or less likely to be violated. For example, if a constraint is a maximum, a change that decreases the constraint or increases the value it constrains makes it more likely that the constraint will be violated. Again, fixes without a clear direction -- e.g., those involving calls to the database or substitutions of symbolic values -- are assumed to have a potentially negative effect on downstream constraint checks.

SALT lists chains of interacting fixes. Each chain originates from a constraint whose tendency to be violated is not affected by fixes for any other constraint and whose own fixes make other constraints more likely to be violated. If fixes for any of the constraints in this second tier make other constraints more likely to be violated, these are added to the chain. Loops are flagged when a constraint recurs in the chain. Knowledge engineers examine the chains and loops in a knowledge base to see whether they represent trouble spots for the standard treatment.

## Diagnosis and Treatment of Trouble Spots

The map displayed by SALT for the HOIST-CABLE-QUANTITY example is shown below. The starting point of the chain is the first constraint, shown between asterisks. There is no possibility that a fix anywhere else will make it more likely that this first constraint will be violated. Constraints are shown without parentheses. All potential fixes for a given constraint are listed below the constraints that they might remedy and indented to the right. They are enclosed in parentheses

<sup>&</sup>lt;sup>4</sup>A related but less serious problem is that a remedy not chosen may have an ameliorating effect on a downstream constraint violation. In such a case, the system might miss a solution in which the total cost of fixing the two violations might be less if a more costly fix were chosen for the first.

along with the direction of change and a number representing the cost of the fix; higher numbers indicate higher cost. Below each fix value is a list of constraints that may be more likely to be violated because of that fix. "LOOP" indicates the occurrence in the chain of a constraint that has already appeared.

```
*
MINIMUM-HOIST-CABLE-QUANTITY
*
(HOIST-CABLE-QUANTITY, Up, 4)
MINIMUM-HOIST-CABLE-QUANTITY
*** LOOP ***
```

This figure displays a loop, but it is not an infinite loop because the system will stop increasing HOIST-CABLE-QUANTITY when the maximum hoist cable quantity is reached. When this happens a dead end will be declared; it is not a premature dead end, however, because it is unavoidable.

A simple case involving a potential premature dead end might look like this:<sup>5</sup>

```
*
MAXIMUM-CAR-JAMB-RETURN
*
(PLATFORM-WIDTH, Down, 10)
(DOOR-OPENING, Substitute, 8)
(OPENING-WIDTH, Up, 6)
MINIMUM-DOOR-OPERATOR-SPACE
(DOOR-MODEL, Up, 4)
```

This map indicates that the less costly change to OPENING-WIDTH, although preferred for its local cost considerations, may cause a violation of MINIMUM-DOOR-OPERATOR-SPACE. The more expensive fixes to PLATFORM-WIDTH and DOOR-OPENING have no negative downstream effects. If selecting OPENING-WIDTH as the fix to implement causes a dead end, this may constitute a premature announcement of failure. Selecting a change to PLATFORM-WIDTH, DOOR-OPENING, or some combination of the three potential changes may remedy the violation to MAXIMUM-CAR-JAMB-RETURN without making it impossible to fix MINIMUM-DOOR-OPERATOR-SPACE.

Currently, the SALT user must hand-code a solution by customizing treatment for the potential site of the dead end. The control shell must be modified so that whenever a dead end is found for a violation of MINIMUM-DOOR-OPERATOR-SPACE, the system will go back and try more expensive fixes at the relevant prior constraint violation, MAXIMUM-CAR-JAMB-RETURN. The SALT map helps the user locate this relevant prior constraint violation.

Below is an example of a potential infinite loop:

<sup>5</sup>This example is completely fictional, since no such trouble spots exist in the current VT knowledge base (Marcus, Stout and McDermott, 1986).

MAXIMUM-TRACTION-RATIO

(CN/T-TO-PLATEOPM-EPONT Down 2)
(COMP-CAREELINIT-WEIGHT Up 4)
(HOIST-CABLE-QUANTITY, UP, 4)
(COMP-CABLE-UNIT-WEIGHT, Down, 4)
MAXIMUM-TRACTION-RATIO
*** LOOP ***
(MACHINE-GROOVE-MODEL, Down, 8)
MAXIMUM-TRACTION-RATIO
*** LOOP ***
(MACHINE-GROOVE-MODEL, Up, 8)
MAXIMUM-MACHINE-GROOVE-PRESSURE
(HOIST-CABLE-QUANTITY, Up, 4)
(COMP-CABLE-UNIT-WEIGHT, Down, 4)
MAXIMUM-TRACTION-RATIO
*** LOÖP ***
(MACHINE-GROOVE-MODEL, Down, 8)
MAXIMUM-TRACTION-RATIO

\*\*\* LOOP \*\*\*

Two of the fixes for MAXIMUM-TRACTION-RATIO can result in violation of MAXIMUM-MACHINE-GROOVE-PRESSURE and in turn, two fixes for MAXIMUM-MACHINE-GROOVE-PRESSURE can loop back to cause violation of MAXIMUM-TRACTION-RATIO. The details of this map reveal that these two potential fixes for each constraint--the changes to COMP-CABLE-UNIT-WEIGHT and MACHINE-GROOVE-MODEL--directly counteract each other. A closer look at SALT's knowledge representation, not in evidence here, indicates that decreasing the CWT-TO-PLATFORM distance to fix MAXIMUM-TRACTION-RATIO does not affect MACHINE-GROOVE-PRESSURE or its maximum. Adding hoist cables to fix MAXIMUM-MACHINE-GROOVE-PRESSURE may actually ameliorate a problem with MAXIMUM-TRACTION-RATIO, although the effect is not substantial enough to warrant its inclusion as a fix for that constraint.

As long as only one of the two constraints is violated, the standard search for a solution based on isolated constraint violations is satisfactory. But if both constraints are violated, the system may thrash. The control shell was modified to treat this latter situation as a special case and investigate fixes for the two constraints in tandem. To do this, one extra piece of information not currently acquired by SALT was needed. If both constraints cannot be remedied at the same time, a VT domain expert would relax MAXIMUM-MACHINE-GROOVE-PRESSURE before violating MAXIMUM-TRACTION-RATIO. Under such a circumstance, the generated system will try to minimize the violation of MAXIMUM-MACHINE-GROOVE-PRESSURE.

Whenever a violation of one of these constraints is detected, the system will check to see if the other has been violated. If it has, the system will reset the values of all potential fix values to their last value before the first violation of either constraint. It will then try out potential fixes in this order, according to whether the fix:

- 1. Helps both.
- 2. Helps one and doesn't hurt the other.
- 3. Helps one but does hurt the other.

In the case of 3, the system will apply the fix to remedy the constraint most important to fix.

# **Future Directions**

The current version of SALT makes good use of its knowledge of how information in a domain-specific knowledge base will be used in constructing a solution. It has already proved useful in developing the VT system. SALT has also been used to acquire a knowledge base in developing an expert system for production scheduling of elevators (Marcus, Caplain, McDermott, and Stout, 1986) and for a portion of the XSEL task of selecting computer components to meet a customer's functional requirements. One major difference between these domains and that of VT is the way in which local fixes interact. When VT decides which proposed fix should be implemented when a constraint has been violated, the major factor determining the "goodness" of a fix is the dollar cost to the company. This dollar cost can be determined independently of the parameters of any particular customer input and in fact is specified by the SALT user before the VT system is ever run. Fixes for one constraint very rarely exacerbate violations of other constraints for VT. In most cases, when this does occur, the system will not thrash but simply increase cost until all options are exhausted.

For the scheduling task, on the other hand, the decision about which order to reschedule or which department must handle rush work depends on such things as what orders are currently scheduled, how close each is to its original expected schedule, and how close departments are to their capacity at the time a reschedule is attempted. This is because fixes for one constraint can very easily exacerbate another constraint violation, and there is the potential for a fix for one constraint to undo a fix for another. This affects the way SALT must elicit knowledge about how to perform schedule revisions.

First, SALT can automate the diagnosis described in the last section by pointing out itself the potential for nonconvergence when it occurs. More importantly, SALT can elicit additional relevant knowledge to address such inadequacies. For example, SALT can suggest preconditions on considering a proposed fix that take into account values likely to cause violations of other constraints. SALT's knowledge representation allows it to locate such values and judge their effect on other values because they are identified by the role they play in problem-solving.

SALT makes a strong commitment to the problem-solving strategy that will be used for the task it will acquire. This allows SALT to represent domain-specific knowledge according to the role it will play in finding a solution for any task that can use this basic strategy. This commitment gives it considerable power in providing guidance to users in the area where they most need it -- making decisions that require consideration of the potential interactions of a single piece of knowledge with everything else in the knowledge base.

# References

Boose, J. 1984. "Personal Construct Theory and the Transfer of Human Expertise," in *Proceedings of the National Conference on Artificial Intelligence*, Austin, Texas.

Brown, D. 1985. "Failure Handling in a Design Expert System," in *Computer-Aided Design*, 17: 436-441.

Clancey, W. 1985. "Heuristic Classification," Artificial Intelligence, 27: 289-350.

Davis, R., and Lenat, D. 1982. Knowledge-Based Systems in Artificial Intelligence, McGraw-Hill.

Kahn, G.; Nowlan, S., and McDermott, J. 1984. "A Foundation for Knowledge Acquisition," in *Proceedings of IEEE Workshop on Principles of Knowledge-Based Systems*, Denver, Colorado.

Kahn, G.; Nowlan, S; and McDermott, J. 1985. "MORE: An Intelligent Knowledge Acquisition Tool," in *Proceedings of the International Joint Conference on Artificial Intelligence* (IJCAI), Los Angeles, pp. 581-584.

Marcus, S.; McDermott, J.; and Wang, T. 1985. "Knowledge Acquisition for Constructive Systems," in *Proceedings of the International Joint Conference on Artificial Intelligence* (IJCAI), pp. 637-639.

Marcus, S.; Stout, J.; and McDermott, J. 1986. "VT: An Expert Elevator Configurer That Uses Knowledge-Based Backtracking," Carnegie-Mellon University Department of Computer Science technical report, forthcoming.

Marcus, S.; Caplain, G.; McDermott, J., and Stout, J. 1986. "Making SALT Generic," Carnegie-Mellon University Department of Computer Science technical report, forthcoming.

Marcus, S., and McDermott, J. 1986. "SALT: A Knowledge Acquisition Tool for Propose-and Revise Systems," Carnegie-Mellon University Department of Computer Science technical report, forthcoming.

McDermott, J. 1986. "Making Expert Systems Explicit," in *Proceedings of the IFIP Congress*, Dublin, Ireland.

Mittal, S., and Araya, A. 1986. "A Knowledge-Based Framework for Design," in Proceedings of the National Conference on Artificial Intelligence, Philadelphia, PA.

Stallman, R.M., and Sussman, G.J. 1977. "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," *Artificial Intelligence*, 9, 135-196.

Stefik, M. 1981a. "Planning With Constraints (MOLGEN: Part 1)," Artificial Intelligence, 16,111-140.

Stefik, M. 1981b. "Planning and Meta-Planning (MOLGEN: Part 2)," Artificial Intelligence, 16, 141-170.

Sussman, G.J., and Steele, G.L., Jr. 1980. "CONSTRAINTS -- A Language for Expressing Almost-Hierarchical Descriptions," Artificial Intelligence, 14, 1-39.

Van de Brug, A.; Bachant, J.; and McDermott, J. 1986. "The Taming of R1," *IEEE-Expert*, forthcoming.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

Measurement of Declarative and Procedural Knowledge in the Development

of a Knowledge Based Media Planning System

Andrew A. Mitchell, University of Toronto

In this paper, we describe our experiences in measuring declarative and procedural knowledge for the development of a knowledge based media planning system. The purpose of the system is to use the information from the annual Marketing Plan and other pertinent information for a particular brand (e.g. Tide detergent) to develop a media plan for the brand. The resulting media plan will provide an allocation of the advertising budget by media (e.g. magazine) and media vehicle (e.g. Time Magazine) over a one year planning horizon.

The planning system will contain two stages. At the first stage, the critical elements of the Marketing Plan will be used to form goals and constraints for the media plan. Examples of the goals will include reach and frequency goals and an allocation of the budget by media and over the planning horizon. Examples of constraints would be the need to use print media for a coupon promotion or the need for a minimum number of Gross Rating Points in a television schedule during a particular time period. The second stage will be the development of the media plan. The development of this plan involves satisfying the multiple goals and constraints. If a feasible media plan cannot be found, then the system will adjust the goals and constraints set at the first stage. The resulting system will provide an interactive environment for the media planner. It will ask the media planner for information at various stages of the process and provide justification and the tradeoffs involved for the various decisions that the system makes.

In developing the system, media planners from a number of major advertising agencies and firms in the United States are being interviewed. Consequently, rather than simply trying to mimic the problem solving of a single expert, we are attempting to incorporate the skills of a number of experts. Based on our current findings it appears that the planning process is similar across media planners, however, there are some differences across media planners in how they use information at different stages of the process. Therefore, one important aspect of the project involves determining the similarities and differences in the process across media planners.

The knowledge acquisition process contains two phases. In the first phase, a number of media planners were interviewed in depth to understand how goals are developed and, then, how these goals are used to develop a media plan. At this phase, media planners were asked to sort a number of media problems into categories with similar goals to understand the goal formation process and were also given a series of media problems with differing goals and asked to develop a media plan for each problem. Think aloud protocols were taken for both tasks. In the second phase, standardized materials will be developed to measure differences between media planners in the development of goals and the media plan. Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# **INFORM:** An Architecture for Expert-Directed Knowledge Acquisition

Eric A. Moore Schlumberger/Applicon 4251 Plymouth Rd Ann Arbor, MI 48105 (313) 995 -6262

Alice M. Agogino Department of Mechanical Engineering University of California, Berkeley 5136 Etcheverry Hall Berkeley, CA 94720 aagogino@euler.berkeley.edu (415) 642-6450

## Abstract

This paper presents an architecture for INFORM, a domain independent, expert-directed knowledge acquisition aid for developing knowledge-based systems. The INFORM architecture is based on information requirements and modeling approaches derived from both decision analysis and knowledge engineering. It emphasizes accommodating cycles of creative and analytic modeling activity and the assessment and representation of aggregates of information holistically represent domain expertise for heuristic classification problem solving, [Clancey85], especially in domains with diagnosis or decision making under uncertainty. Influence diagrams are used as the knowledge structure and computational representation. We present here a set of information and performance requirements for expert-directed knowledge acquisition, and describe a synthesis of approaches for supporting the knowledge engineering activity. We discuss potential applications of INFORM as a knowledge engineering aid, specifically as an aid for developing insight about the encoding domain on the part of its user.

## 1. Introduction

Hindrances to widespread application of expert systems include what are typically significant allocations of resources, of critical personnel (the expert) and of knowledge engineering effort and equipment. The knowledge engineer's efforts to replicate the knowledge underlying expert performance through encoding techniques that maintain the form of that knowledge are known as knowledge acquisition; the design of tools and techniques to manage and support the process, as well as the active guidance of the process, is known as knowledge engineering. Knowledge acquisition is by far the hardest and most time consuming part of the expert systems building problem.

"Knowledge acquisition bottleneck" understates the significance of the effort required to assess from a domain expert the information necessary to achieve expert performance. The resources required to build an expert system seem to have funneled the application of knowledge-based technology to only high payoff projects, involving only experts with highly valued skills. Here, the more rarified the expertise, and the more significant the application, the harder it is for someone outside of the expert's domain -- the knowledge engineer -- to build a system to replicate it. "Knowledge acquisition Klein Bottle" might be more appropriate.

31-()

This work was performed at the University of California, Berkeley, and was funded in part through grants from the National Science Foundation PYI Program, University of California's Project MICRO, General Electric, and IBM.

What can one do? We could relax the performance requirement, and settle for a knowledge-based system without expert performance. But this is not the most effective use of organizational resources. We could find persons with familiarity or proficiency with both knowledge engineering tools and representation and the encoding domain [Fox83], but even these persons are a scarce organizational resource. We can find less skilled persons in the domain that are likely to be more articulate about their problem solving [Dreyfus80], but there is no assurance that these people will share their conceptual structure of the domain with the expert. Perhaps we can eliminate the expert knowledge engineer, and look for a way to let the expert encode directly.

The thought of having experts encode their expertise is compelling. Without an intermediary between the expert and the system, there is no noise introduced to the encoded knowledge, there is no time spent for the knowledge engineer to learn the language and concepts of the domain, and the resultant system has the expert's -- and not an intermediary's -- view of the domain [Friedland81]. For this, one risks losing process efficiency, for the expert must understand the knowledge representation and learn how to use the tool, one risks a potential loss of transparency, if the expert must recast his thinking in the tool's terms, and one risks the potential failure to objectively and fundamentally address the expert's reasoning in the domain.

Like many established engineering organizations, the U. C. Berkeley Mechanical Engineering Department has many potential research applications for knowledge-based technology, rich areas of domain expertise, and many senior and articulate experts, but lacks readily available organizational knowledge engineering expertise or tools demonstrably appropriate for the potential applications.

There is a very strong motivation to develop not a just a toolkit, but a procedural aid that will allow, for example, Master's level engineering students to successfully and efficiently employ knowledge engineering techniques and technology for practical problem solving. Ongoing research has produced IDES, the <u>Influence Diagram-Based Expert System</u>, for doing probabilistic inference and planning using influence diagrams [Agogino85, Agogino86] and [Agogino86...]. This paper presents an architecture for INFORM, (INFluence diagram FORMer), an expert-directed knowledge acquisition aid and interface for building knowledge-based systems in IDES.

## 2. Prior Work on Knowledge Acquisition

We draw a distinction between *techniques*, *tools*, and *aids*. A technique is a set of procedures, heuristics, or guidelines for performing Knowledge Acquisition (KA) or Knowledge Engineering (KE). A tool provides software support for application of the techniques, but no guidance on its own; knowledge engineers use tools. An aid is a tool that provides process guidance on its own. An expert undertaking any phase of a knowledge engineering project requires an aid.

KE tools, techniques, and aids in the literature address different areas of the knowledge engineering process: *encoding context*, the phase of determining how the characteristics of the domain, the expert, the user, and the application will affect or constrain KA procedure; *knowledge structuring*, the process resulting in an initial description of the knowledge base in the computational representation; and *knowledge refinement*, the process of model focusing and validation. Our research focus here is not so much on tools for KA, but on techniques and aids for KE.

Early work in KE was concentrated on developing tools and representations. The concept of the "knowledge level", [Newell82] seeking to formally describe domain knowledge and problem solving at a level independent of implementation, has influenced the development of ontological representations of different problem solving domains, [Clancey85, Alexander86] and languages specialized to problem solving types [Bylander86]. Recent emphasis has been on methodologies for structured KA, [Freiling85, Kline86] and [deGreef85] on assuring that the KA process meet the requirements of the application's organization, [Grover83] and on aids for rule refinement [Kahn85, Eshelman86] and [Ginsberg85, Langlotz86].

The organizational and structured knowledge acquisition approaches are information-driven in the sense that they are a formalism, a set of activities, which produce documents and assure that information requirements and checks are met. These approaches emphasize building a paper knowledge base, or building a conceptual or knowledge level structure of the domain before committing programming resources; here, experts can describe their domain structure, in some accessible representation freed from the implementation representation and with minimized reformulation by the KE. Here, however, the KE is later involved in rule encoding and refinement.

De Greef and Breuker [deGreef85] see two basic approaches to knowledge engineering: the skills/programming-based rapid prototype and test approach [Hayes-Roth83, Brownston85] and the structured knowledge engineering approach, which guides and supports knowledge acquisition while implementation is deferred. INFORM actually falls between the two; we employ model refinement techniques from decision analysis and knowledge engineering in an environment that is predominantly structured knowledge acquisition.

Successful KA aids for domain dependent systems in both KE and Decision Analysis (DA) exist; often their design provides a domain-based encoding language or domain specific graphics, or some superset of domain concepts from which the temporal encoding problem will be identified [Holtzman85, Differding84] and [Merkhofer79].

"Domain independent" means that, for a given problem solving approach, the user must create the concepts, rather than select them, or that all meta-models of domains are included in the tool model. Two aids for knowledge structuring, ETS [Boose84, Boose85] and ROGET [Bennett83] elicit the expert's structure of domain concepts though sequences of comparisons among sets of proposed objects. Both are intended for use by domain experts and result in "executable" rule bases. ROGET aids the user in choosing the appropriate inference technique and ontological representation, given information about the user's experience and the problem solving type (as subsets of the classification problem solving model). INFORM is intended to be domain independent, but instantiable to specific domains.

## 3. Expert-Directed Knowledge Acquisition

The notion of "expert systems" is the desire to replicate an expert's problem solving performance in a domain. While expert systems are proving to be effective computational representations of knowledge and expertise, they are not complete (some would argue not even adequate) cognitive models of that knowledge or expertise. So the process of capturing knowledge, of transferring the expert's cognitive structures, representations, and methods to computational domain structures, knowledge representations, and procedures, will almost certainly entail its reformulation. If, for the expert, the act of articulating this knowledge to an audience is novel, then the expert is also reformulating this knowledge. Knowledge engineering is both a descriptive and creative modeling activity.

We view knowledge engineering as a model design and software engineering activity. A proportionally small amount of KE time is actually spent programming [Freiling85, Grover83]; much of the skills (and effort) of knowledge engineering are modeling skills -- analysis and reduction, information management, and process decision making -- as well as the traditional emphasis on performance replication through incremental refinement.

An expert-directed KA interface must support all of these activities to in turn successfully support a model's initial elicitation and eventual refinement.

The key assumptions behind any approach to self-encoding are that

- it is plausible that the expert can efficiently use the encoding interface, that the expert understands how to use the tool, the problem, and is motivated enough to use it as it is intended to be used;
- the expert can think abstractly about the domain and problem solving within it, i.e. identifying variables and influences;
- that a structured, analytic approach to thinking about one's domain knowledge and problem solving can achieve a refinable model;
- the inevitable loss of transparency in encoded information is acceptable if the expert can somehow assure the performance of the model or if the expert is capable of thinking in the terms of the transformed model.

## 4. Decision Analysis, Influence Diagrams, and Knowledge-Based Systems

Decision analysis (DA) brings a body of experience to structured KA that meshes well with other approaches from within the AI community.

### 4.1. Decision Analysis

The decision analysis cycle [Matheson77] is an iterative and interactive proscription for assuring that essential steps in the decision process or decision-making problem have been taken. It separates the process into deterministic structuring, probabilistic assessment, and informational phases. Assessment and modeling procedures direct the formation of choices, information, and preferences into the decision set.

Both practitioners of DA and KE face the problem of attention focusing, not in making analyses complicated enough to be comprehensive, but rather keeping them simple enough to be affordable and useful [Howard80]. DA structuring aids have taken a largely "top down" approach to modeling a domain, and the KE aids a "bottom up" approach to describing the relations in a domain.

### 4.2. Influence Diagrams

Influence diagrams [Miller76, Rege86] are an attractive conceptual and operational representation for domain expertise. We use influence diagrams as a knowledge structure: a way of organizing knowledge that is operational, but makes no cognitive claim, though they are intuitively appealing. Influence diagrams have developed into a decision analysis tool that graphically represents the structure of the decision problem but maintains the computational utility of the decision tree [Schachter85]. They are a three-layered knowledge representation, consisting of information at three hierarchical levels: relational, functional, and numerical. This hierarchy accommodates well the way people tend to model from simple to complex, and from conceptual to numeric.

At the relational level, influence diagrams are directed acyclic graphs that represent the interdependence of uncertain events in a complex system. Nodes represent sets of possible events, or a range of properties for some object. The presence of an arc indicates the possibility that the outcomes of one node are somehow influenced by the outcomes of the other. At the relational level, they superficially resemble semantic nets and frames. A major distinction is that Bayes' theorem allows topological solution, or "re-orienting" of influence diagrams. Pearl's work with Bayesian inference nets [Pearl85] uses inheritance in a frame based system to propagate uncertainties in a structure that closely resembles influence diagrams, though without decision nodes.

The functional level is a specification of the type of relationship between nodes, or "how" an particular event or object influences another. The functional level is traditionally probabilistic, with quantitative relations compressed into the stochastic ones, but influence diagrams can readily accommodate fuzzy, logical, and other functional relations [Rege86......]. The numerical level is a quantitative measure of the "extent" of the relationship.

Figure (1) describes a diagnostician's model for a simple centrifugal pump. At the relational level, we can say that the pump's "discharge" is influenced by the "foot valve state" and "strainer state". The likelihood that discharge is high, low, or nil, is influenced by the likelihood that the foot valve is open or closed and the likelihood that the strainer is clear, partially clogged, or clogged. At the relational level, we can specify that the arc from foot valve to discharge is "logical"; if the foot valve is closed, the discharge is nil. Or we could specify a probabilistic relation, and give a distribution on the probability of discharge being high, low, or nil, given some joint distribution of strainer and foot valve states. The diagnostic inference problem is formulated as, for example, "given some flow meter reading X, and some pressure gauge reading Y, what is the probability that the strainer is clogged?"



Figure 1: Sample Influence Diagram

# 4.3. Bringing Decision Analysis to Knowledge-Based Systems

INFORM, because it is based on influence diagrams, is seen as best fitting applications under the heuristic classification definition. [Clancey85]. The formal influence diagram representation is quite concise; there are nodes (a set of possible states for an event), states spanning the range of possible outcomes or values for the event, and probabilities on the occurrance of those states conditioned on other events. An arc in an influence diagram represents a heuristic link between a class of concepts in the domain. Data abstraction is subjective, rather than symbolic; the information lies in the uncertainty assessment or from further structuring, rather than in endorsements or in classification hierarchies.

Applying DA to knowledge-based systems means that we focus on designing problem solving models that effectively replicate expert performance, rather than concentrating on implementing descriptions of that performance. It is important to separate replication of performance from duplication of procedure-- at best, duplication is unlikely to result in performance improvement.

Langlotz, Shortliffe, and Fagan [Langlotz86] point out one of the side benefits of doing first and second order sensitivity analysis with heuristics: the KE has to think more broadly about the concept, not just what its value is, but what it could be, and how likely those other value might be. DA is normally employed for signifigant non-routine decision-making where there is uncertainty about the state of the factors influencing the decision, the outcomes of the decision, or the extent to which the factors may influence the outcome. Knowledge-based systems are normally restricted to important but routine problem solving, perhaps with the most frequency to deterministic classification style problems. In situations where it is uneconomical or impossible to replicate the expert problem solving processes, the DA approach may be a viable way to approximate expert performance without explicitly relying on the processes behind it.

Barr argues for knowledge-based systems that provide insights, and not merely answers [Barr85]. He sees the largest measure of the utility of expert systems in the fact that their construction forces critical re-evaluation of one's own expertise. The same has been said of Decision Analysis [Howard80].

Non-transparency (reformulation, rather than replication of a true expert's problem solving skills), represents a potential corruption of those skills, but can improve domain skills in non-experts, persons who would not ordinarily get the benefit of the KE's critical attention. Non-experts and experts alike may gain improvements through articulating, structuring, and recording for examination relationships and strategies in the problem solving domain.

Broadly, AI/Knowledge-Based Systems techniques and technology offer:

- information manipulation & management
- passive and intelligent interface design
- models of users and user actions
- prototyping/system development techniques
- tools and techniques for model refinement
- techniques for heuristic control

and Decision Analysis techniques offer:

- normative models for decision making
- practical encoding techniques for uncertainty
- robust techniques for encoding structure
- experience in organizational integration & acceptance

We see particular appeal in bringing the top-down modeling and Bayesian uncertainty approaches of Decision Analysis and the influence diagram conceptual and computational knowledge structure together with the software engineering tools and performance refinement techniques of Knowledge-Based Systems.

## 5. The Information Requirements for Expert-Directed Knowledge Acquisition

"Data driven" design has confounded many an interface. The design of an interface must be based upon the needs and abilities of the set of users for the set of tasks required for the application. However, the interface must still assure that it gets to the application the data it needs to run. We view INFORM here as a port for putting information into a program -- subject to requirements for content, quality, and efficiency of expression.

There is no escaping the need to engineer information in order to represent knowledge. At issue here, of course, is how to best give the expert some responsibility for knowledge engineering. INFORM is responsible not only for meeting the information needs of the computational knowledge representation, the influence diagram, but for meeting the information needs of a knowledge engineering process: context structuring, refinement, and process control.

There are three basic types of information INFORM must use:

Model:	the knowledge base
Procedural:	information revolving around the state, history, and direction of
	the KE process
Insightful:	information adjunct to analytic and creative thinking about and
Ū	explanation of the model

In this section, we discuss these information types in terms of their form and their assessment.

### 5.1. Model Information

The information in the knowledge base is divided into Computational, Structural, and Uncertain conceptual information types.

Computational- These are the requirements of the formal influence diagram and IDES. Nodes, states, probabilities, outcomes, arcs map from a heterogeneous collection of "C" data structures to matrices

and probability distribution for topological transformation and numeric calculation within IDES.

Structural- Influence diagrams, augmented with context and assumption tags, and with their graphical representation, are an appealing way to structure the knowledge in a domain.

An influence diagram represents a model of a problem. In the context of assessment, it is a framework for experimenting with that model's behavior. The encoded diagram must represent information and must communicate an understanding. Much of this deeper information is descriptive: representing controlling assumptions, constraints on those assumptions and endorsement for or against them, intentions, histories, and alternatives. To communicate this understanding, we must represent information of different types: graph information, text, numerical, deterministic, logical, and uncertain.

Uncertainty- Despite the naturalness of the influence diagram representation, both temporal and domain acquisition problems are difficult for an expert or some other user to solve without experience or training and in some cases, without assistance. While Bayesian probability is a particular strength of this knowledge representation, encoding it presents difficulty. It is tedious. People's numeric estimates of uncertainty invariably do not accurately represent their underlying judgement without some structured revision and debiasing [Raiffa70, Kahneman82]. The process of encoding uncertain information may affect the values assessed and so is critical to the utility of that information [Spetzler77].

The many alternative uncertainty calculi are in part responses to these problems. The failure, however, of any one representation to win widespread acceptance as the "best" underscores the need for richer representations.

Bonissone and Tong [Bonissone85] present further guidelines for assessment; these dovetail with what we already know to be important in terms of the structuring the uncertain variable in decision analysis. Their discussion is good because it presents the uncertainty encoding activity explicitly as an information problem. So then, for each piece of evidence, one should determine the:

- measure of certainty/uncertainty
- source of the evidence
- credibility of the source
- environmental conditions under which the source gathered information
- sensitivity of the goal to evidence
- cost of facility to gather information
- likelihood of succeeding in gathering information
- cost of this information gathering task
- default plan to accomplish this task

The encoding of expert's uncertainty estimates is as least as important as the internal representation of that uncertainty in a knowledge base. One essential perspective on uncertainty representation which sometimes gets lost is that the representation must be intuitively agreeable to the expert – both the expert and the uncertainty representation must speak the same language. As Bonisonne [Bonissone85...] points out, it will ultimately take a mix of verbal and numeric representations to adequately cover the Babel of uncertainty representations used by different experts in different domains.

Influence diagrams are founded on Bayesian probability. Cheeseman [Cheeseman85] argues that Bayesian probability, if properly used, can be worked around virtually all objections to it; in his view, the faults of Bayesian probability are based primarily on the misperceptions of its critics. On the other hand, a number is a rather sterile representation of a quantity that, cognitively, appears to be in large part verbal [Zimmer85]. A strictly Bayesian numeric estimate is very convenient, and axiomatically correct, but is often misleading without a complete view of the priors implicit in the assessment. Further, a single number overestimates the crispness of the state of knowledge about that uncertainty. A verbal assessment incorporates more factors than a numeric one, but computation, without loss of information, requires that the user's fuzzy functions be known as a context-specific mapping of verbal to possibilistic [Zimmer85] and probabilistic numeric distributions. Evidential reasoning emphasizes articulating priors acting on an estimate and the decision-making power of simply ranking outcomes (much like the Analytic Hierarchy Process in decision-making [Saaty80]). All of these approaches,

under some conditions, make a strong case for themselves. With influence diagrams, we are committed to representations that can ultimately be mapped to Bayesian probability.

Certainly a judgement on the strength or weakness of one representation or another should consider encodability of that representation. In assessing an uncertainty estimate, considering all the approaches, one would want to:

- rank comparable outcomes in order of likelihood
- assess a verbal (qualitative) estimate
- assess numeric Bayesian values
- elicit underlying evidence for an assessment
- estimate the range of uncertainty

But decision analytic information assessment calls for no less that all of this information. What we find is that, even though the computational representation may be considerably sparer, the conceptual representation must include an aggregate of information about the uncertain quantity.

We contend that a well designed encoding and representation environment can make the encoding of Bayesian probabilities for expert systems less forbidding and more accurate. Such an environment would support a composite conceptual representation of uncertainty (including linguistic, numeric, underlying and conditioning priors), a mapping from verbal to numeric, and from numeric to verbal, and a numeric Bayesian calculus.

The approach we will take is straightforward:

- first assess reference linguistic distributions in a broad context
- use these linguistic assessments as a "first pass"
- for refinement, with more sensitive variables, or for variables misleadingly represented by the linguistic assessment,
- refine the linguistic assessment in a more specific context or employ traditional numeric encoding techniques

## **5.2.** Procedural Information

Supporting the KA process, for a self-encoding user, or for some combination of KE and expert, means

- information management tools (for information programming)
- advice and tools for making procedural decisions.

Many KA tools provide programming support, support interpreted incremental refinement, provide rule prompters, or a rule compilers based on a rule language. The DA framework is an approach that consolidates or would amenably consolidate all of these approaches.

The most ambitious approaches would interact in a natural language directly with the user without the user understanding the representation within the system. One view [Reboh81] favors a system that requires the collaboration of the KE, but with techniques and tools for support of critical phases requiring little KE training. Such a system would in effect re-distribute portions of the KE's expertise between the support tools and a domain expert or less skilled KE. This view is at the heart of the INFORM approach.

## 5.3. Information for Insight

We regard "insight" as the the creation and revision of a mental picture of the domain and processes within it, and the recognition and evaluation of possibilities and tradeoffs inside it. The modeling interface should provide the information and techniques for developing and maintaining insight about the model. We are of course concerned with conveying timely information from the model to the user, the other problem is easing and organizing the conveyance of appropriate information from user to model. The base issue is achieving a cognitive fit between user and interface. INFORM must provide a familiar medium and acceptable stimulus for modeling effectively.

Essay writing is a metaphor for the "progressive formalization" [Holtzman85] of a decision analytic model. A contention of the INFORM approach is that expository writing could accomplish (in effect) what the motivation and (in part) structuring phases are intend to accomplish in the probability encoding process [Spetzler77]: give the user the opportunity to address critically the assumptions, intentions, and methods behind the model.

The "standard form" for writing an essay is a well known and widely taught framework for expository thinking and discourse. As an encoding approach, it goes a step beyond simply writing rules into some subset of a natural or domain language. It puts relations into paragraphs and sections in support of a problem statement or thesis-- the user thinks about components of the model in an structured sense, and in the context of the whole model.

In reviewing a knowledge base encoded through INFORM, the DA, expert, user, or KE is in a sense reading a story; from the contents of the KB, the reader is supposed to be able to piece together a problem solving "narrative". Applications to be used with persons other than the encoding expert require explanations that are dependent on both the line of reasoning and the model description; a very critical element for KA is visualizing an audience for the application. Viewed this way, we can say that the "rules of journalism" apply here too. The reader must know — and the interface must somehow assess: Who, What, When, Where, Why, & How for each concept and relation.

## 6. INFORM

## 6.1. Design Goals

These three performance goals capture the essence of what INFORM is intended to accomplish:

- (1) sufficiency, getting the encoded information right in terms of the influence diagram representation;
- (2) correctness, avoiding and correcting conscious and unconscious misrepresentations of expert judgement; and
- (3) providing insight, at minimum representing the correct information in a comprehendable form, at best completely capturing an expert's underlying model of the objects, relationships, and inferences in his domain.

## 6.2. Architecture

There are four conceptual levels to the INFORM architecture. They build from satisfying information requirements to giving more tools and advice for insight and finally to system which effectively tutors its user through the KE process.

The first level is to fill the diagram to sufficiency, through satisfying structural and computational constraints. The second is to employ the kinds of feedback that decision analysts and knowledge engineers employ: diagram drawing on demand, graphic feedback on distributions, ordinal, deterministic, and stochastic and sensitivity analysis, comprehendable interaction language, and opportunity for and access to extensive explanation about the process and the encoded information. The third is the "heuristic" approach, where the system provides hints and suggestions for encoding to the user based upon normative models of the encoding process and sticky points in the domain. Finally, with "expert aid", the system provides aid (hints, requests for explanation, reformulation) based upon encoding session information, normative models of the encoding process, and descriptive models of the problem domain and user's encoding style.

The INFORM user, in an encoding session, goes through a problem and session structuring module, and a succession of editing and analysis phases. At any stage, the user may get help about the syntax, options, or intent of the current phase, comment about some aspect of the model or the modeling process, or review the some graphic or textual aspect of the model. Figure (2) shows the main modules and editor sequence.



Figure 2.: Process Paths and Modules in the INFORM Architecture

## 6.3. Knowledge Structuring

There are two key ideas to INFORM's guiding structuring refinement approach:

(1) Start modeling at the lowest level of precision or specificity

(2) Increase specificity only for the best improvements in model performance.

The user is free to edit the model, and accept or reject advice on what task to choose next, but is guided through model analysis and refinement.

In the relational editor, the user specifies combinations of node name, node label, node description and arcs. On exiting the editor, this information is parsed and "incomplete information" is identified; the user is prompted to to provide, for example, a description for a node identified only by a label.

The nodes operated on in the relational editor determine the nodes to be operated on in subsequent phases, in the functional, and numerical editors. The analysis phases are directed by ordered lists: do sensitivity analysis with the nodes the user is least confident about, expand the nodes that the outcome is most sensitive to, assess in a different way those uncertainty estimates the user is least (or most confident) about. The order of nodes operated on is determined by user ranking, or by the rank of the node within the influence diagram. Below are some of the activities in the INFORM architecture.

Set Context	system application? encoding goal? identify user?
	calibrate linguistic uncertainty?

"might do this next"

estimate modeling effort?

edit and compile nodes and arcs

potential modifications? encoding plan?

edit and compile functional form, states name, label, description, explanation, plan

for each: name, label, description, givens, explanation

consider reducing the number of states in these nodes

check for cycles, bushyness, sort objects by importance?

Model at Relational Level

- Describe the Model:
- Assure Completeness:
- Look for Insight:
- Offer Analysis:
- Offer Advice:

Revise Model?

Model at Functional Level

- Describe
- Assure Completeness:
- Offer Analysis:
- Offer Advice:

Revise Model?

Model at Numeric Level

- Describe
- Assure Completeness:
- Offer Analysis:
- Offer Advice:

Revise Model?

assess aggregate uncertainty information sensitivity, performance analysis "focus next on these most sensitive nodes"...

## **INFORM** Encoding Activities

#### 6.4. Knowledge refinement in INFORM

"Refinement", in the DA context, is directed towards attention focusing, typically through ranking, and deterministic and stochastic sensitivity analysis, and towards balancing the modeling effort in terms of both structural granularity and value of additional modeling effort. Refinement in rule-based expert systems building is a process of rule addition and modification leading ultimately to performance replication. Performance improvement in an knowledge-based system generally comes with more specificity granularity; because of the large assessment effort behind properly encoding probabilities, a good decision model will expand and contract through each refinement cycle. With influence diagrams, there is tradeoff in the granularity of the uncertainty representation and that of the model structure. Formally, influence diagrams rely on implicitly incorporating conditioning factors within the uncertainty assessment and in the concept's definition to result in a polished but condensed model. Rule-based expert systems representations, on the other hand, force this contextual information to be explicitly expressed as rules. Part of the refinement process in INFORM is the successive elaboration of what the model represents.

The success of our approach to encoding uncertainty during refinement is contingent on at least three things: given no new information, some consistency of uncertainty verbal to numerical mappings is maintained over time and domain; the success of linguistic revision given new contextual information; and the extent to which simulation/feedback response is incorporated into revised estimates.

## **6.5.** INFORM in the Future

The super-structure for INFORM has been implemented in C, and presently a single display text & graph editor assesses relational information and automatically generates complex influence diagrams. We have written a linguistic calibration program; work is continuing at UC Berkeley with experiments to sample uncertainty vocabularies of graduate and undergraduate engineering students. Work will continue on with the re-implementation and complete development of the INFORM architecture at Schlumberger/Applicon with the Strobe/Impulse object programming and knowledge base editing tools [Smith86, Smith83].

The principal advantage of influence diagrams over decision trees is the explicit graphic representation of the interdependencies of events. Influence diagrams are fundamentally graphic entities; once a diagram has been drawn initially, the interface too should be organized graphically. Because we want to simultaneously represent different types of information about the model and the modeling process, a single view is inadequate. The interface under development will have static windows -- for model graphics, model text, prompt window, editor/comments and "pop-up" windows -- for agendas, menus, advisors, and uncertainty encoding and representation. Given the need for a graphic representation, the interface should allow the diagrams to be created graphically (to be drawn on the screen), in addition to generating the graph from the user's entries of nodes and arcs.

INFORM is intended to be domain-independent, but within the architecture, it is easy to add to the interface the kind of checking rules that allow for domain and user dependent meta-knowledge about encoding. RACHEL [Holtzman85] is one such system, a domain dependent intelligent decision support aid for infertile couples.

Effective modelling approaches rely in part on the underlying domain and in part on the modeller cognitive style. INFORM is a system intended to replace at least in part the expertise of the KE in directing and in giving advice to the encoder about the KE process, and in representing the encoded information. Implementation of the architecture may ultimately support active modeling and guidance of an expert's encoding effort. Requirements for an INFORM tutor would be:

- measurable or deducible standards of knowledge engineering performance and methodology,
- and that these measures are conditionable on a fairly small set of inferable or directly assessable measures.

### 6.6. Evaluation

INFORM should be tested for absolute performance, as a interface and as a modeling tool, and for relative performance, against an encoding expert. The real test is to take an encoding problem from scratch with an acknowledged expert, and try to build a working system. The typical test of comparing system performance against that of an encoding expert is inappropriate at this point but ought to be an eventuality. At the interface level, we have used and continue to use "good design" checklists [Heckel84], but that is no real assurance of a good interface without testing and experience. At this stage, while we are still developing an "integrated" system, user comments and ad hoc evaluation are especially useful. More traditional approaches to interface evaluation, based upon a number of user's (with expertise/skill) exposure with a common problem, will be the next step. It will be interesting to test people with measurably different levels of expertise, to look for correlations between level of skill and self-encoding ability, and to test, for example, an undergraduate design class, with some diagnostic problem based on their machine design project.

A testing issue that is separable from the evaluation of the entire interface is the effectiveness and accuracy of our linguistic uncertainty encoding approach. Testing areas of interest include looking at differences in the language of uncertainty between estimates about uncertain events from inside and outside the encoding domain, consistency of judgements between subdomains, and the efficacy of different approaches to representing the encoding problem and conducting refinements.

3(-1)

## 6.7. Potential Uses of INFORM

When can we use a stand-alone aid like INFORM for constructing a knowledge-based system or decision support system? We can divide the spectrum of application problems down into significant and insignificant problems. Significant problems are "high stakes" problems. For significant problems, in an novel domain, one may still be able to use INFORM as a preprocessor (as in ETS) for initiating the model, but would certainly expect DA and KE involvement in model refinement and validation. Significant problems in a stable and well understood context are liable to see the involvement of DA and KE, but it may not be necessary. In both cases, INFORM must accommodate well the involvement of DA and KE. For problems whose solution is robust, or where solution precision is not critical, INFORM is plausibly stand-alone: that is the minimal design goal.

Other potential roles for INFORM in KE:

- structure the domain for explanation, as a component of an intensive KA process;
- help the novice KE work with the expert;
- as a personal KA tool for "small" problems;
- domain dependent aid for "large" problems.

## 7. Conclusion

We feel knowledge acquisition aids must support information assessment and presentation and must provide support for undergoing a sound modeling process. Fundamentally, the INFORM architecture is an aid for building models; it draws its knowledge structure and modeling approach from Decision Analysis, and it approach to handling information and heuristics about encoding from Knowledge Engineering. It is well suited for classification problem solving, especially under uncertainty. The support INFORM will provide for experts encoding is as a top down design aid, focusing on descriptions of the domain concepts and structure, rather than on examples of problem solving in the domain. Structure is edited, rather than induced. Such direct involvement of the expert in constructing an operational model of the domain we feel will aid *knowledge engineering for insight*, aiding the development of expert behavior not only on the part of the system, but on the part of the encoder as well.

## References

#### Agogino85.

A. M. Agogino, "Use of Probabilistic Influence in Diagnostic Expert Systems," in *Proceedings* of the 1985 ASME International Computers in Mechanical Engineering, Boston, MA, vol. 2, pp. 305-310, Aug 4-8 1985.

#### Agogino86.

A. M. Agogino and A. Rege, "IDES: Influence Diagram Based Expert System," in Mathematical Modelling in Science and Technology Proceedings of the Fifth International Conference on Mathematical Modelling, July 29-31 1985, University of California, Berkeley, ed. X.J.R. Avula, Pergammon Press, 1986.

### Agogino86.

A. M. Agogino and A. Rege, "Sensor Integrated Expert System for Manufacturing and Process Diagnostics," Proceedings of the Symposium on Knowledge-Based Systems, ASME Winter Annual Meeting, Anahiem, CA, December 7-12 1986.

### Alexander86.

J. H. Alexander and M. J. Freiling, J. L. Staley, S. Rehfuss, and S. L. Messick, "Knowledge Level Engineering: Ontological Analysis," *Proceedings American Association for Artificial Intelligence '86*, pp. 963-968, 1986.

#### Barr85.

A. Barr, "Systems That Don't Understand," Cognitiva: Artificial Intelligence and Neuroscience, Paris, June 1985. Bennett83.

J. S. Bennett, ROGET: A knowledge-based consultant for acquiring the conceptual structure of an expert system, Stanford HPP-83-24, October 1983.

## Bonissone85.

P. P. Bonissone, Summarizing Uncertain Information With Aggregation Operators (draft), General Electric Corporate Research and Development, New York, March 11, 1985.

### Bonissone85.

P. P. Bonissone and R. M. Tong, "Editorial: Reasoning with Uncertainty in Expert Systems," International Journal of Man-Machine Studies, vol. 22, pp. 241-250, 1985.

## Boose84.

J. H. Boose, Personal Construct Theory and the Transfer of Human Expertise, pp. 27-33, Proceedings American Association for Artificial Intelligence '84, 1984.

#### Boose85.

J. H. Boose, "A Knowledge Acquisition Program for Expert Systems Based Upon Personal Construct Psychology," International Journal of Man-Machine Studies, vol. 23, pp. 495-429, 1985.

Brownston85.

L. Brownston, R. Farrell, E. Kant, and N. Martin, Programming Expert Systems in OPS5, Addison-Wesley Publishing Company, 1985.

## Bylander86.

T. Bylander and S. Mittal, "CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling," The Al Magazine, pp. 66-77, August 1986.

## Cheeseman85.

P. Cheeseman, "In Defense of Probability," Proceedings International Joint Conference on Artificial Intelligence '85, 1985.

#### Clancey85.

W. J. Clancey, "Heuristic Classification," Artificial Intelligence, vol. 27, pp. 289-350, 1985.

Differding84.

J. C. Differding, The OPAL Interface: General Overview, Stanford Oncocin Projects, 1984.

## Dreyfus80.

S. E. Dreyfus and H. L. Dreyfus, "A five stage model of the mental activities involved in directed skill acquisition," ORC 80-2, U. C. Berkeley Industrial Engineering and Operations Research Department, February 1980.

## Eshelman86.

L. Eshelman and J. McDermott, "MOLE: A Knolwedge Acquisition Tool that Uses Its Head," *Proceedings American Association for Artificial Intelligence '86*, pp. 950-955, 1986.

Fox83.

M. S. Fox, S. Lowenfeld, and P. Kleinosky, "Techniques for Sensor- Based Diagnosis," *Proceedings International Joint Conference on Artificial Intelligence* '83, pp. 158-163, 1983.

## Freiling85.

'M. Freiling, J. Alexander, S. Messick, S. Rehfuss, and S. Shulman, "Starting a Knowledge Engineering Project: A Step by Step Approach," *The AI Magazine*, pp. 150-163, Fall, 1985.

## Friedland81.

P. Friedland, "Acquisition of Procedural Knowledge from Domain Experts," in Proceedings International Joint Conference on Artificial Intelligence '81, pp. 856-861, 1981.

### Ginsberg85.

A. Ginsberg, S. Weiss, and P. Politakis, "SEEK2: A Generalized Approach to Automatic Knowledge Base Refinement," *Proceeedings International Joint Conference on Artificial Intelligence* '85, pp. 367-375, 1985.

## Grover83.

M. D. Grover, "A Pragmatic Knowledge Acquisition Methodology," Proceedings International Joint Conference on Artificial Intelligence '83, pp. 436-438, 1983.

### Hayes-Roth83.

F. Hayes-Roth and D. A. Waterman, "An Overview of Expert Systems, Chapter 1," in *Building Expert Systems*, ed. F., D. A. Waterman, D. R. Lenat (eds.) Hayes-Roth, Addison-Wesley Publishing Company, 1983.

#### Heckel84.

P. Heckel, The Elements of Freindly Software Design, Warner Books, 1984.

## Holtzman85.

S. Holtzman, Intelligent Decision Systems, Phd thesis, Stanford Engineering-Economic Systems, reprinted by Strategic Decisions Group, March 1985.

#### Howard80.

R. A. Howard, "An Assessment of Decision Analysis," Operations Research, vol. 28, no. 1, pp. 4-27, Jan/Feb 1980.

#### Kahn85.

G. Kahn, S. Nowlan, and J. McDermott, MORE: An Intelligent Knowledge Acquisition Tool, pp. 582-584, Proceedings International Joint Conference on Artificial Intelligence '85, 1985.

# Kahneman82.

D. Kahneman, P. Slovic, and A. Tversky, Judgement Under Uncertainty: Heuristics and Biases, Cambridge University Press, 1982.

### Kline86.

P. J. Kline and S. B. Dolins, "Problem Features that Influence the Design of Expert Systems," *Proceedings American Association for Artificial Intelligence* '86, pp. 956-962, 1986.

#### Langlotz86.

C. P. Langlotz, E. H. Shortliffe, and L. M. Fagan, "Using Decision Theory to Justify Heuristics," *Proceedings American Association for Artificial Intelligence* '86, pp. 215-219, 1986.

#### Matheson77.

J. E. Matheson and R. H. Howard, "Introduction to Decision Analysis," in *Readings In Decision Analysis*, Strategic Decisions Group, 1977.

### Merkhofer79.

M. W. Merkhofer, B. Robinson, and R. J. Korsan, "A Computer Aided Decision Structuring Process," Tech. Report 7320, SRI, Menlo Park, CA, June 1979.

#### Miller76.

A. C. Miller, M. W. Merkhofer, and R. H. Howard, Development of Automated Aids for Decision Analysis, SRI, Menlo Park, CA, 1976.

#### Newell82.

A. Newell, "The Knowledge Level," Artificial Intelligence, vol. 18, pp. 87-127, 1982.

#### Pearl85.

J. Pearl, "Fusion, Propagation, and Structuring in Bayesian Networks," in Workshop on Probability and Uncertainty in Artificial Intelligence, UCLA, August 14-16, 1985, RCA and AAAI, 1985.

#### Raiffa70.

H. Raiffa, Decision Analysis: Introductory Lectures on Choices Under Uncertainty, Addison-Wesley Publishing Company, July 1970.

#### Reboh81.

R. Reboh, Knowledge Engineering Techniques and Tools for Expert Systems, Linkoping Studies in Science and Technology No. 71, Software Systems Research Center, Linkoping University, S-581 83, Linkoping, Sweden, 1981. Rege86.

A. Rege, "Representing and Solving the Probabilistic Inference Problem in Expert Systems," UC Berkeley Expert Systems Lab Working Paper, August 1986.

Rege86.

A. Rege and A. Agogino, "Fuzzy Influence Diagrams," UC Berkeley Expert Systems Lab Working Paper 06-86-01, June 1986.

Saaty80.

T. L. Saaty, The Analytic Hierarchy Process, McGraw-Hill, New York, 1980.

Schachter85.

R. D. Schachter, "Intelligent Probabilistic Inference," in Workshop on Probability and Uncertainty in Artificial Intelligence, UCLA, August 14-16, 1985, RCA and AAAI, 1985.

Smith83.

R. G. Smith, "Strobe: Support for Structured Object Knowledge Representation," Proceedings International Joint Conference on Artificial Intelligence '83, pp. 855-858, 1983.

Smith86.

R. G. Smith, R. Dinitz, and P. Barth, "Impulse-86: A Substrate for Object-Oriented Interface Design," Proceedings of the ACM Conference on Object Oriented Programming Systems, Languages, and Applications, 1986.

Spetzler77.

C S. Spetzler and C-A. S. Stael von Holstein, "Probability Encoding in Decision Analysis," in *Readings in Decision Analysis*, Decision Analysis Group, Stanford Research Institute., Menlo Park, CA, 1977.

Zimmer85.

A. C. Zimmer, "The Estimation of Subjective Probabilities via Catagorical Judgements of Uncertainty," in Workshop on Probability and Uncertainty in Artificial Intelligence, UCLA, August 14-16, 1985, RCA and AAAI, 1985.

deGreef85.

P. deGreef and J. Breuker, "A Case Study in Structured Knowledge Acquisition," Proceeedings International Joint Conference on Artificial Intelligence '85, pp. 390-392, 1985.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

## ACQUIRING DOMAIN MODELS

Katharina Morik \*

Technische Universität Berlin Institut für Angewandte Informatik Computergestützte Informationssysteme Projekt KIT-Lerner, Sekr. FR 5-8 Franklinstr. 28/29 D-1000 Berlin 10 West-Germany

<ciskit@db0tuill.bitnet>

#### Abstract

Whereas a Learning Apprentice System stresses the generation and refinement of shallow rules of a performance program, presupposing a domain theory, BLIP<sup>1</sup> is mainly concerned with the construction of a domain theory as the first phase of the knowledge acquisition process. In this paper the BLIP approach to machine learning is described. The system design is presented and the already implemented knowledge sources are shown with their formalisms and functions for the learning process.

### 1 Introduction

Following from the growing interest in the application of expert systems machine learning is now viewed as a means for building up and refining the knowledge base of an expert system for a particular domain and for a particular consultation class. This puts new requirements on machine learning and has led to a discontent with the former inductive learning paradigm which can be characterized by a two-step procedure consisting of a training phase and a performance phase. In the training phase, a teacher has to present several examples, each classified as positive or negative, forming a complete set with regard to the concept to be learned. This set must be free of

<sup>\*</sup> The KIT-LERNER project is partially supported by the German Ministry for Research and Technology (BMFT) under contract ITW8501B1. Industrial partners are Nixdorf Computer AG and Stollmann GmbH.

<sup>&</sup>lt;sup>1</sup> BLIP stands for 'Berlin Learning by Induction Program'.

contradictions. The ordering of examples perhaps is important. In the worst case, all the system does is to abstract away features, thus acquiring a general concept. As far as the knowledge acquisition for an expert system is concerned, this procedure is, of course, unsatisfactory. Instead, real world observations should be exploited as examples (Rajamoney et al. 85). These can be a user's problem solving acticities (e.g. Brazdil 78, Mitchell et al. 85, Smith et al. 85), visual data of objects (e.g. Phelps et al. 86), or facts, as is the case with BLIP. No training mode should be necessary and no trainee should be in demand.

The new paradigms of machine learning, namely explanation-based learning, analytic learning, verification based learning, deductive learning have overcome the limitations of the learning-from-example paradigm. However, there is still another constraint on these learning approaches: A complete domain theory is presupposed. Furthermore, in the case of deductive and verification based learning, the domain theory must be such that the correctness of each rule can be shown with certainty. Thus the problem is to build up a consistent domain theory. The standard criticism of inductive learning techniques is that finding the right descriptive terms for the description language is the real learning task. This argument also applies to the new learning paradigms in a modified way: building up the domain model is the real problem. The first phase of knowledge acquisition is the hardest problem of the entire learning process (van Someren 86). This problem has been largely ignored in existing machine learning approaches. BLIP is an attempt to cope with this problem.

Of course, learning cannot start without any knowledge. One way out of the dilemma is to start knowledge acquisition with just two types of knowledge:

- a "sloppy" domain model entered by the user,
- general structures that reflect structures of human thought, rather than structures of the world (Anderson 75).

In knowledge representation systems for domain modeling (e.g. KL-ONE and its derivates) a so-called knowledge engineer has to structure the domain, and define the relevant objects, their properties and relations between them. A learning program, however, should not require the user to do all this. An

initial sloppy model of the domain should suffice, a well organized model should be the result of the learning process. The learning process itself should be guided by domain-independent, general knowledge that reflects the human ability to adopt to new situations. This general knowledge can be incorporated into the system and, thus, need not be entered by the knowledge engineer.

In this paper, the system BLIP is described whose task is to aid in the construction of a domain model. The learning process is guided by metaknowledge which can be viewed as the representation of cognitive structures that ease the knowledge acquisition process.

## 2 BLIP's approach to machine learning

BLIP is the learning part of an overall system called LERNER<sup>2</sup>. The performance element of the overall system is the expert system shell TWAICE<sup>3</sup>. BLIP requires the user to specify a sloppy domain model by

- defining predicates and the sorts of their possible arguments
- entering facts about the domain expressed with those predicates

BLIP then discovers properties of those predicates and establishes relations between them thus structuring the domain. An example of the user's entries and the program's output is presented below. Finally, the domain model is transformed into the TWAICE-format rules (for the overall LERNER architecture see Emde et al. 86). The rule part of the expert system that represents knowledge about a particular domain is then filled. Beyond the range of BLIP is the acquisition of problem solving knowledge which elsewhere has been called 'domain principle' knowledge (Swartout 83) and which is acquired by systems like SEEK.2 (Ginsberg et al. 85), LEAP (Mitchell et. al 85) or others (cf. Steels et al. 85).

<sup>&</sup>lt;sup>2</sup> BLIP is implemented in MPROLOG and is fully operational under VM370 on an IBM4381. The learning algorithms are based on the METAXA.2 system //Emde 84//.

<sup>&</sup>lt;sup>3</sup> TWAICE is a registered trademark of Nixdorf Computer AG. It is in the framework of EMYCIN.

To learn without bearing a particular consultation system in mind has several advantages. First, the learned domain model may be used more than once to construct several expert systems. Suppose BLIP has acquired a domain model of the 'side-effects-of-drugs world'. This knowledge may be used to build an expert system which assists a user in choosing a drug with a small number of side effects for a known complaint. But the knowledge may also be used to build an expert system for an agency which has to assess the risks of new drugs. The domain model that is necessary to solve these problems is by and large the same, but the problems differ in the domain principle knowledge that is necessary.

The same domain model may also be used to build an expert system which differs from a previously built system only in the number of questions that will be asked, e.g., because more information is available from the users. In such cases where even the domain principle knowledge is the same, the difference lies in the consultation knowledge (cf. Clancey 79). The term 'consultation knowledge' is applied to knowledge about the goal of the consultation with an expert system (e.g., 'Determine a fault in a computer system'), the knowledge about the data that have to be asked from the user, the knowledge about default assumptions that should be used if the user is not able to answer a question, and other dialog handling routines. The implications of choosing another consultation class are so important that expert systems are built from scratch even if expert systems similar with respect to domain principles and domain model already exist.

Second, as different kinds of knowledge are acquired and represented seperately it is possible to explain the expert system behavior to the user with methods described by Swartout (Swartout 83).

Before we explain the learning process, let us have a look at the knowledge sources. First we will describe the constructions for sloppy modeling, then the meta-knowledge sources.

### 2.1 BLIP's Knowledge Sources

Knowledge representation in BLIP is not homogeneous. Although the representation formalisms are all in the spirit of predicate logics (of higher order if meta-knowledge is involved), each knowledge source has its own format. This lack of uniformity can lead to redundant representations. However, the functionality of the knowledge sources and their efficient use was of particular importance to us.

Facts are represented declaratively as predicates with arguments, a representation that is natural to the user. We assume a sorted logic. The argument sort masks are stored in the lexicon of predicates (LEXICON). The superconcept relation can be defined for two argument sorts but need not be. Excerpts of the knowledge sources can be seen in figure 1.

Analogous to the sort restrictions for the arguments of a predicate there is a kind of sort restriction for the arguments of a meta-predicate, too. That is, the predicates are sorted in a tree structure and each layer of the tree is named. This tree-structure is stored in the knowledge source LAYERLEX. For an example, see a simplified diagram of the LAYERLEX for our pharmacy domain (Fig.2).

These two knowledge-sources are domain-dependent. The argument types of predicates for a certain domain model are to be entered into LEXICON. The predicate layers can be declared in order to explicitly describe the effect of a certain consultation class on a domain model, here, the recommendation class as applied to the pharmacy domain. This is done filling the LAYERLEX.

For building up a sloppy model of side-effects of drugs, the knowledge engineer defines, for instance, the following predicates:

- CURE\_1 (<drug>, <illness>)

- CURE\_2 (<substance>, <symptom>)
- CURE\_3 (<drug>, <symptom>)
- CURE\_4 (<substance>, <illness>)
- CONTAIN (<drug>, <substance>)
- STRENGTHEN\_l (<drug>, <illness>)
- STRENGTHEN\_2 (<substance>, <symptom>)
- STRENGTHEN\_3 (<drug>, <symptom>)
- MONO-SUBSTANCE (<drug>)

RULE-SCHEMATA: LEXICON: reductive (p1,s2,s): lexicon (strengthen 2, p1(x1, y1) => s2(x1).substance, symptom). opposite-1 (op1,op2,s): lexicon (subsort, op1(x1) => not (op2(x1)).pain, symptom). opposite-2 (p1,r1,s): p1(x1, y1) => not (r1(x1, y1)).STRUCTURE : LAYERLEX: (reductive (p1,s2,s)) layerlex (strengthen 2, lexikon (p1, sort1, sort2) & disadvantage, wf). lexikon (s2, sort1) layerlex (is-good-for, layerlex (p1,m1,11) & recommendation, bw2). layerlex (m1,mm1,12) & layerlex (s2, x, 12)). CHARACTERISTIC SITUATION: cspos (reductive(p1,s2,s), p1(x1, y1), s2(x1)).csneg (reductive(p1,s2,s), p1(x1,y1),not (s2(x1))). cspos (opposite-1(op1,op2,s), op1(x1) & not op2(x1). csneg (opposite-1(op1,op2,s), op1(x1) & op2(x1)). FACTS: RULES: mono-substance (aspirin). opposite-1(op1,op2,s) => not inclusive-1(op1,op2,s). multi-substance (novalgin). mono-substance(x1) => not opposite-1 (mono-substance, multi-substance(x1). multi-substance, all). opposite=2 (opposite=1, inclusive-1, all).

Fig. 1: Knowledge sources of BLIP

- MULTI-SUBSTANCE (<drug>)

- ADVANTAGE (<patient>, <drug>)

- IS\_GOOD\_FOR (<patient>, <drug>)

- DISADVANTAGE (<patient>, <drug>)



Fig.2: LAYERLEX for the pharmacy domain

Further predicates can be defined by demand. The user is not forced to give the semantics of these predicates by explaining their relations. The relations which s/he might have in mind are to be made explicit by the machine. For example the following rules have been learned by BLIP:

- If a drug contains a substance, then its negative effects which are expressed by EFFECT, STRENGTHEN, DISADVANTAGE, and ADDICTION are to be inherited by the drug.
- If a drug is of advantage and not of disadvantagefor a patient, then it is good for the patient.
- If a drug is a multi-substance, then it is not a mono-substance.

These rules have to be represented explicitly in a domain model. For a user, however, they are so "natural" that s/he might not want to write them down. In the BLIP framework, just after entering the predicate definitions the user can express facts. For example:

- ENTAIL (aetz, paracetamol) - ENTAIL (aetz, ass)

- ENTAIL (aspirin, ass)
- STRENGTHEN\_2 (ass, stomach-ache)
- STRENGTHEN\_2 (paracetamol, kidney-trouble)
- STRENGTHEN\_3 (aetz, kidney-trouble)
- STRENGTHEN\_3 (aspirin, stomach-ache)

The user enters these facts just taking them from a text book on drugs. S/he needs not take care of the order of the facts or even of a clear definition of the predicates. BLIP then comes up with rules like those mentioned above. In BLIP's notation, a part of the first rule is written as follows:

ENTAILS (x,y) & STRENGTHEN\_2  $(y,z) \Rightarrow$  STRENGTHEN\_3 (x,z)

This rule corresponds to a meta-fact. In the next paragraph BLIP's metaknowledge is explained.

### 2.2 Meta-knowledge in BLIP

Meta-knowledge plays the central role in our approach to knowledge acquisition. Meta-knowledge is used for

- checking consistency

- inducing rules from facts (the learning process)

- deducing rules from other rules.

Meta-predicates express relations between and attributes of predicates. Metameta-predicates express relations between meta-predicates. A distinction of types between meta-predicates and meta-meta-predicates is not made. A metapredicate can be used at every meta-level. It is used to represent the relation between concepts of the level directly below.

#### Example:

factl: MONO-SUBSTANCE (aspirin)

fact2: MULTI-SUBSTANCE (novalgin)

fact3: OPPOSITE 1 (mono-substance, multi-substance)

fact4: OPPOSITE\_2 (opposite\_1, inclusive\_1)

Fact3 is at the meta-level with respect to fact1 and fact2, it is at the object-level with respect to fact4. A meta-predicate, e.g. OPPOSITE, can be
used at every meta-level; it can be a meta-predicate, a meta-meta-predicate, a meta-meta-predicate, and so on.

Meta-knowledge in BLIP does not refer to a rule format or a knowledge representation formalism as does the meta-knowledge of AM, EURISKO, TEIRESIAS (Davis, Lenat 82). Nor does it refer to control knowledge or run-time behavior as described in (Hayes-Roth et al. 83). Meta-knowledge in BLIP refers to predicate constants, describing properties or relations of certain properties or relations.

A meta-predicate expresses a structural relation that is given by a RULE-SCHEMA. For example, the rule-schema for OPPOSITE\_l is<sup>4</sup>:

$$P(x) \Rightarrow not Q(x)$$
  
 $Q(x) \Rightarrow not P(x)$ 

and for OPPOSITE\_2:

$$P(x,y) \Rightarrow not Q(x,y)$$
  
 $Q(x,y) \Rightarrow not P(x,y)$ 

The rule-schema for INCLUSIVE\_1 is:

 $P(x) \Rightarrow Q(x)$ 

The rule-schema for EQUIVALENT is:

$$P(x) \Rightarrow Q(x)$$
  
 $Q(x) \Rightarrow P(x)$ 

It is easily seen that P or Q can be meta-predicates and that relations between meta-predicates can be expressed by meta-meta-predicates using the very same rule-schemata.

The restrictions on the applicability of meta-predicates to predicates are given by the knowledge source STRUCTURE. A meta-predicate can be applied to predicates if the argument structures of the predicates fulfill certain restrictions and the predicates are of the appropriate types. For example, the meta-predicate OPPOSITE can only be applied to two predicates of the same layer which have the same argument restrictions, because direct opposition of

<sup>4</sup> in another notation:

 $\forall$  P,Q,x: OPPOSITE\_1(P(x),Q(x)) <=> (P(x) => not Q(x))

relations presupposes that the relations are comparable. Another function is to exclude antinomies: The distinction of types is obeyed, i.e. predicates of the first type are seperated from those of higher types by the structural descriptions.

If a meta-predicate holds for a particular predicate this is represented in two ways. BLIP employs as well a procedural (inferential) representation as a declarative representation of the same information. This is, as Weyhrauch (Weyhrauch 80) puts it, a consequence of the procedural-declarative debate in AI. For example, the meta-fact

W\_TRANSITIVE (contain, cure\_2, cure\_3)

expresses in a declarative way the same information as the rule

 $CONTAIN(x,y) \& CURE_2(y,z) \Rightarrow CURE_3(x,z)$ 

On the one hand, a meta-fact is stored in the fact part of the inference engine. On the other hand, a rule is generated automatically for the meta-fact and stored in the rule part of the inference engine. The two representations have different functions. With the help of the rule new facts are inferred and thus the factual knowledge of the system is enhanced. For example, given

CONTAIN (novopirin, ass), CURE\_2 (ass, head-ache),

the new fact

CURE\_3 (novopirin, head-ache)

can be entered into the knowledge base. With the help of the declarative representation other meta-facts are inferred in the following way: when there is a meta-meta-fact whose corresponding representation as a rule has, e.g., W\_TRANSITIVE as a premise and some other meta-predicates in its conclusion, these meta-predicates will be considered true with the argument bindings of the meta-fact concerning w\_transitivity without testing. Also, negated meta-predicates as the conclusion of a meta-meta-fact are useful because the corresponding meta-facts then need not be tested.

For example:

i) meta-meta-fact: OPPOSITE\_2 (opposite\_l, inclusive\_l)
ii) meta-fact: OPPOSITE\_1 (mono-substance, multi-substance)
iii) meta-fact: INCLUSIVE\_1(mono-substance, multi-substance)

Meta-fact iii) need not be tested if ii) is known because of the rule which corresponds to meta-fact i):

i') meta-rule: OPPOSITE\_l(p,q) => not INCLUSIVE\_l (p,q)

In our example, p is MONO-SUBSTANCE and q is MULTI-SUBSTANCE. Only the declarative representation of rules enables us to deduce rules from other rules.

For entering a fact or a rule into the inference engine knowledge about representation format of the inference engine is used (FORMAT).

In order to verify whether or not a meta-predicate holds for certain predicates the criteria for the validity of a meta-fact are represented in such a way that they can be used as pattern for a search process in the factual knowledge of the inference engine. These patterns are called "characteristic situations" (Emde et al. 83). For each meta-predicate patterns for positive and negative situations are stored. The CHARACTERISTIC SITUATIONS are generated out of the rule-schemata automatically.

The main information about meta-predicates is captured by the rule-schemata and the structural descriptions. The characteristic situations could be generated at run-time as part of the testing process as well as the format knowledge could be part of the entering process of the inference engine. But, -as is well known - it is of advantage that most of the knowledge is extracted from processes, because it is then transparent (and particularly in machine learning where the expert user requires control over the system's processing) and easy to change<sup>5</sup>.

<sup>&</sup>lt;sup>5</sup> The interface for the meta-knowledge acquisition and the transducer for the automatic generation of redundant knowledge representations has been implemented by Sabine Thieme.

There is a third knowledge source which could be realized as part of a process: the PRIORITIES of meta-predicates. There is an efficient ordering of meta-predicates to be tested for a predicate. It may be viewed as the knowledge of how to construct an "experiment" in order to prove a hypothesis. This knowledge may be coded in the hypothesis testing or - as we do - is concentrated into a knowledge source. The efficient ordering of hypotheses depends on the meta-meta-facts. The ordering itself is represented in a network-like knowledge source, which can be used for choosing the hypotheses which are to be tested next.

#### 2.3 The Learning Process

The main components of the learning process are

- the generation of hypotheses,
- the rating of hypotheses and
- the testing of hypotheses.

The architecture of BLIP is shown in figure 3. The components interact via an agenda. The hypotheses are generated syntactically by taking advantage of the structural descriptions. For each predicate all possible meta-facts which take that predicate as an argument are constructed.

The rating component attachs a value between 0 and 1000 to each such hypothesis. There are several criteria for rating:

- efficiency:

the hypothesis is easy to test, or the hypothesis would exclude a lot of other hypotheses (this is the priority knowledge);

- activeness:

former tests of the particular hypothesis could not be accomplished with a clear result;

- effectiveness:

the number of stored facts which correspond to the predicate(s) for which a meta-predicate is to be tested

Meta-facts which require no testing are immediately generated and entered into the inference engine. They always have the highest rating.



## Fig. 3: system architecture of BLIP

A hypothesis is tested by searching in the fact part of the inference engine for facts matching the characteristic situations of this hypothesis. The number of positive and negative examples are counted. If there are only positive examples and no negative one, and the number of positive examples exceeds a threshold, then the hypothesis is verified, the corresponding meta-

fact is entered into the factual part, and the corresponding rule is generated and entered in the rule part of the inference engine. If sufficiently many negative situations found, but no positive ones, then the hypothesis is clearly falsified. If neither the number of positive nor the number of negative examples exceeds the threshold, nothing is entered into the inference engine, but the activeness of the hypothesis increases.

The difficult case is, of course, if facts support a hypothesis, but there is at least one exception. We prepared SUPPORT SETs for this case. Each metafact has as one argument the support set for which the meta-fact is valid. The default assumption is that the support set is ALL (s. fig. 1). But if exceptions occur, these are described and excluded from the support set. The description is given by a condition which is introduced as a supplementary premise in the corresponding rule. The condition can be very simple, e.g. the rule is not valid for a certain concept. If more is known about the exceptions the condition can become more complicated, e.g. the rule is not valid for all concepts with a certain attribute or for all concepts with a certain relation to other concepts. The exceptions are analyzed searching for statements which are true for all of them. In this way new concepts can be introduced.

Meta-knowledge also assists in the manual acquisition of knowledge. Instead of typing in facts and rules for each object of the domain, meta-facts may be entered into the system. From these meta-facts rules are generated and from these facts are deduced. Checking sets of rules for consistency is a problem which is dealt with in, e.g. (Suwa et al.82), (Keller et al. 81). Our approach of using declaratively represented consistency conditions saves theorem proving and allows for repairing inconsistencies easily. Consistency of meta-facts is checked with the help of meta-meta-facts. Consistency of facts is checked with the help of meta-facts.

### 3 Further Research

The processes described above will be implemented on a SYMBOLICS 3670 in PROLOG. The main research topics will be:

- improving and reorganizing the predicates and their typed arguments

- developing the approach of the support sets further, so that the knowledge base can automatically be structured according to relevant subdomains
- handling exceptions, noisy data, and evidence for the inappropriateness of the model built so far

If a meta-predicate cannot apply to a predicate only because of the order of the arguments, the order should be rearranged. Similarly, predicates with n arguments can be split into r predicates with m arguments (m < n). Of course, there may conflicts occur. Criteria are the needed in order to decide which rearrangement is the most efficient.

If support sets or the exception sets become more complicated they can be used to structure the domain model. The model may be divided into sub-domains, in which certain relations hold These sub-domains may be interrelated. If it were possible to construct a taxonomy of sub-domains, this could reflect the overall structure of the particular domain and could be used as overview knowledge. An approach in that direction has been made by Emde et al. (83). Considering the taxonomy as the optimal "Gestalt" of a theory provides a guide as to how the sub-domains should be organized. It allows for improving even a consistent domain model (DeKleer at al. 81).

Handling noisy data is another (although interrelated) topic. There are some problems with this. First, classifying facts requires a theory. But this theory is not fully known before the system has accomplished its task and no new facts are to be expected. There is no teacher who classifies the facts for the system as a "deus ex machina". Therefore, the system has to use the rules generated so far, which may be an incomplete or even false theory.

Second, improving a theory means not only to increase the amount of rules, but also to modify it. This may include the deletion of rules and those facts which have been deduced with the help of these rules (belief revision), or even the refutation of the theory and building up an alternative, more appropriate theory. Another modification of a theory is to introduce "invisible facts" by the help of which most of the rules and inferred facts can be kept but the counter-examples are also covered.

Third, building up a body of interrelated inference rules would become impossible, if each new incoming fact had the chance to change the former

generated rules. The system were too busy to revise, its behavior would become oscillating.

It will be one of our prior tasks to operationalize criteria for the decision whether a fact should be rejected, a rule should be deleted with the dependent facts, or the theory should be refuted. Some ideas of how deal with evidence for the inappropriateness of a model are presented in Emde (86).

#### **4** REFERENCES

Anderson, B.F. (75): "Cognitive Psychology" Academic Press

- Brazdil, P. (78): "Experimental Learning Model" in: Proceedings of AISB/GI, Hamburg, pp.46
- Clancey, W.J.(79): "Tutoring rules for guiding a case method dialogue"; The International Journal of Man-Machine Studies 11, p.25-49, 1979
- Davis, R./Lenat, D.B.(82): "Knowledge-Based Systems in Artificial Intelligence" McGraw-Hill
- DeKleer, J./Brown, J.S.(81): "Mental Models of Physical Mechanisms and Their Acquisition"; in: J.R.Anderson (ed.): Cognitive Skills and Their Acquisition, Lawrence Erlbaum, 1981, p.285-309
- Emde, W./ Habel, Ch./ Rollinger, C.-R.(83): "The Discovery of the Equator (or Concept Driven Learning)"; In: Proc. IJCAI-83, Karlsruhe, 1983
- Emde, W. (84): "Inkrementelles Lernen mit heuristisch generierten Modellen"; KIT-Report 22, Technische Universität Berlin, 1984
- Emde, W. (86): "Great Flood in the Blocks World (or Non-cumulative Learning)"; Proceedings of the European Workshop Session on Learning, Paris-Orsay, 1986
- Emde, W./ Morik, K.(86): "The BLIP System" KIT-Report 32, Technische Universität Berlin, 1986
- Ginsberg, A./Weiss, Sh.(85): "Seek2: A Generalized Approach to Automatic Knowledge Base Refinement"; In: Proc. IJCAI-85, p.367-374, 1985
- Hayes-Roth, F./Waterman, D.A./Lenat, D.(83): "Building Expert Systems" Addison Wesley, 1983
- Keller, R.M./ Nagel, D.(81): "Some Experiments in Abstraction of Relational Characteristics"; Technical Report DCS-TM-15; Rutgers University, May 1981
- Mitchell, T.M./ Mahadevan, S./Steinberg, L.I. (85): "LEAP: A Learning Apprentice for VLSI Design"; In: Proc. IJCAI-85, 1985
- Phelps, B./Musgrove, P. (86): "Representation and Clustering in Conceptual Learning" Proceedings of the European Workshop Session on Learning, Paris-Orsay, 1986
- Rajamoney, S./DeJong, G./Faltings, B. (85): "Towards a Model of Conceptual Knowledge Acquisition through Directed Experimentation"; In: Proc IJCAI-85, p.688-690, 1985

- Smith, R.G./Mitchell,T.M./Winston,H.A./Buchanan, B.G. (85): "Representation and Use of Explicit Justification for Knowledge Refinement"; In: Proc. IJCAI-85, p.673-680, 1985
- Steels, L./van de Velde, W. (85): "Learning in Second Generation Expert Systems"; In: J.S. Kowalik (ed.): Knowledge-based Problem Solving, Prentice-Hall Inc., New Jersey, 1985
- Suwa, M./Scott, A.C./Shortliffe, E.H.(82): "An Approach to Verifying Completeness and Consistency in a Rule-Based Expert-System", Stanford University, Stanford, CA, Report No. STAN-CS-82-922, 1982
- Swartout, W.R.(83): "XPLAIN: A System for Creating and Explaining Expert Consulting Programs"; In: Artificial Intelligence 21, p. 285-325, 1983
- van Someren, M.W.(86): "Constructive Induction Rules: Reducing the Description Space for Rule Learning", Proceedings of the European Workshop Session on Learning, Paris-Orsay, 1986
- Weyhrauch, R.W.(80): "Prolegomena to a Theory of Mechanized Formal Reasoning" in: Artificial Intelligence 13, pp.133, 1980

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

## USING A DOMAIN MODEL TO DRIVE AN INTERACTIVE KNOWLEDGE EDITING TOOL

## Mark A. Musen, Lawrence M. Fagan, David M. Combs, Edward H. Shortliffe Medical Computer Science Group Knowledge Systems Laboratory Stanford University School of Medicine Stanford, California 94305-5479 U.S.A.

The manner in which a knowledge acquisition tool displays the contents of a knowledge base affects the way users interact with the system. Previous tools have incorporated semantics that allow knowledge to be edited either in terms of the structural representation of the knowledge or the general problem-solving strategy in which the knowledge is ultimately used. An alternative paradigm is to use the semantics of the application domain itself to govern access to an expert system's knowledge base. This approach has been explored in a program called OPAL, which allows medical specialists working alone to enter and review cancer treatment plans for use by an expert system called ONCOCIN. Knowledge acquisition tools based on strong domain models should be useful in application areas that are highly structured and for which there is a need for repetitive knowledge entry.

# INTRODUCTION

Knowledge acquisition for expert systems can be a laborious process because of difficulties in communication between the knowledge engineers who build the systems and the domain experts whose knowledge is to be represented in the computer. Knowledge engineers typically lack the background needed to pose optimal questions about a particular application area. At the same time, domain experts may find it hard to introspect on their problem-solving strategy and often have little appreciation for how knowledge engineers formalize expertise as a knowledge base. While the use of structured interviewing techniques (de Greef and Breuker, 1985) and protocol analysis (Fox *et al.*, 1985) has begun to define an established methodology for knowledge engineering, often many cycles of programming, system evaluation, and re-programming are still necessary. It has long been recognized that the development of new expert systems might be greatly expedited if experts could somehow enter their knowledge directly into computers without relying on knowledge engineers as intermediaries (Davis, 1979).

Although domain experts should not be expected to become programmers or to learn knowledge representation techniques, it is still possible to reduce their dependence on knowledge engineers during construction of expert systems. One approach is to provide the experts with some type of *knowledge editor* to aid in updating and reviewing the contents of the knowledge base. Because the technicalities of encoding knowledge are hidden by the editor, the user can think about the information being entered in more familiar terms. In fact, the more a user can be insulated from the implementation details of representing expert knowledge, the easier it is to view the knowledge intuitively and abstractly. Knowledge editing programs consequently attempt to transform an expert's knowledge between some higher level, external representation entered into the editor and the internal format required by the target expert system.

Every computer program is written with particular semantic assumptions about the data on which it operates. These assumptions, which are reflected in the way users interact with the program, form a *model* of the data. For example, simple text editors generally employ a model in which the data represent characters in a document; such editors accept commands whose semantics relate to modifying characters or lines of text. On the other hand, a knowledge editor such as TEIRESIAS (Davis, 1979) must assume a different sort of model. The data on which TEIRESIAS operates represent rules in the knowledge base of an expert system. Interaction with TEIRESIAS involves identifying erroneous or missing rules and making necessary corrections. Although all of the rules in the knowledge base collectively define a problem-solving strategy, it is important to note that TEIRESIAS does not address the knowledge from that perspective; the system is designed to focus on the individual rules in a single chain of reasoning.

OPAL, the knowledge editor for a cancer therapy management system known as ONCOCIN (Shortliffe et al., 1981), is based on a more abstract kind of data model--that of the structure of the domain knowledge itself. A detailed model of oncology treatment gives OPAL the ability to solicit and display knowledge of cancer treatment plans using graphical "forms" and other visual representations that are intuitively understandable to expert physicians (Musen et al., 1986a). An example of an OPAL "form" is shown in Figure 1. Knowledge specified graphically using OPAL is first stored internally in an intermediate representation and then automatically converted to a format used by ONCOCIN to provide clinical consultations. OPAL's use of a detailed model of the domain minimizes dependence on knowledge engineers by allowing entry of new treatment plans directly by physicians.

This paper examines the use of domain models as the foundation for interactive knowledge acquisition tools with which expertise can be entered independent of the particular representation required by the target expert system. We show that a domain model which includes strong assumptions about the application area can serve to accelerate knowledge engineering significantly. Because we base our discussion on our experience with OPAL, it is first necessary to describe ONCOCIN and its area of application.

# THE PERFORMANCE ELEMENT: ONCOCIN

Optimal therapy for most cancers is not known. In academic medical centers, patients are often enrolled in clinical trials that study the efficacy of alternative treatment plans for given types of cancer. The written descriptions of these complex treatments, called *protocols*, contain extremely detailed specifications for how patients with particular tumors should be managed by their physicians. Protocols dictate combinations of drugs (called *chemotherapies*) that are given in a specific order over time. Required laboratory tests and radiation treatments (if any) are also specified. The complexity of oncology protocols can make it difficult for physicians to remember precisely what treatments may be required in certain circumstances. The written protocol descriptions may also have areas of vagueness or ambiguity that may be difficult for practitioners to resolve (Musen *et al.*, 1986b). Cancer therapy is thus an excellent domain for the application of expert systems technology.

ONCOCIN uses a knowledge base of cancer protocols to make therapy recommendations. Physicians enter required data into the system each time a patient is seen in clinic. ONCOCIN then uses its knowledge of the protocol in which the patient is enrolled to determine appropriate treatment given the clinical information supplied by the physician. (If the physician should ultimately disagree with the program's suggested therapy, he or she is free to modify ONCOCIN's recommendation.)

ONCOCIN's task can be viewed as a problem in skeletal plan refinement (Friedland and Iwasaki, 1985). The program identifies the appropriate protocol for a patient and constructs a treatment plan by selecting and instantiating plan elements at successively greater levels of detail. For example, ONCOCIN might determine that the plan for a

Chemotherapy	VAM	_
Days in Chemotherapy	21	
Choose Subcycle		
Drug		
VP-16	ADRIAMYCIN	METHOTREXATE
Specify Dose	Specify Dose	Specify Dose

#### Figure 1: Sample OPAL "Form"

This table, which has been reproduced directly from the computer screen, displays information concerning the drug combination "VAM", consisting of VP-16, Adriamycin, and methotrexate. For each specified drug, OPAL creates a separate object in the intermediate representation of the ONCOCIN knowledge base. These drug objects are linked to the object for VAM chemotherapy, thus defining their "context".

particular patient visit should be initiation of a new drug combination (chemotherapy). The general plan to begin a chemotherapy is then refined into separate plans for giving each of the component drugs. Next, the dose and route of administration (*e.g.*, intravenous or oral) for each drug must be determined.

The ONCOCIN knowledge base is encoded heterogeneously using three basic representations. First, a hierarchy of *frames* encoded using an object-oriented language (Lane, 1986) defines the various structural entities for each protocol in the knowledge base (Figure 2). Each treatment plan has a unique *protocol* object that identifies other objects representing the constituent radiation treatments and chemotherapies. The *chemotherapy* objects in turn specify objects that describe the component drugs.

*Production rules*, the second form of knowledge representation in ONCOCIN, are linked to each object in the planning hierarchy. The rules are invoked during skeletal plan refinement to conclude the values of special variables called *parameters*, establishing such concepts as the clinical condition of the patient, interpretations for various laboratory tests, and required adjustments in drug dosages (Shortliffe *et al.*, 1981). Associating rules with specific objects representing the components of the planning task explicitly defines the context within which each rule is applicable, enhancing the efficiency of the consultation system. For example, the rules linked to an object representing a particular chemotherapy conclude the values of parameters related just to that combination of drugs.

The third form of knowledge representation concerns encoding the sequences of chemotherapies and radiotherapies to administer to the patient over time. This procedural knowledge is represented in finite state tables. Each finite state table and the name of the ONCOCIN parameter whose value reflects the "current" patient state is stored in an object called a generator (Musen et al., 1986c).

Defining knowledge of a new protocol for ONCOCIN requires (1) creating an object hierarchy describing the component elements, (2) linking appropriate production rules (and the parameters they conclude) to the various objects, and (3) specifying the temporal sequence of chemotherapies and radiation treatments in terms of a finite state table. Each of these representational issues is handled transparently by OPAL. Because OPAL uses a strong model of the oncology domain as its data model, users can think about protocol knowledge independently of how that knowledge is ultimately encoded in the expert system.



Figure 2: Hierarchy of Objects in ONCOCIN

Both the ONCOCIN knowledge base and the intermediate knowledge representation used by OPAL maintain a hierarchy of objects that reflects the components of each protocol's treatment plan. A protocol is comprised of specific radiation treatments and chemotherapies. The chemotherapies are made up of various drugs. In the knowledge base used by ONCOCIN, production rules are linked to each object. The consultation is conducted by traversing the object hierarchy and invoking appropriate rules.

# THE DOMAIN MODEL IN OPAL

Analysis of a domain at the "knowledge-level", irrespective of whatever notations might be used to *encode* such knowledge, has been suggested as an important first step in the development of expert systems (Newell, 1982). Once the intended actions, goals, and vocabulary of the application are understood, a system of symbols can then be used to represent the knowledge. However, the particular choice of symbols is in some ways arbitrary.

OPAL's domain model has been derived from a knowledge-level understanding of therapy planning in oncology. Review of dozens of chemotherapy protocols, in addition to experience in building a prototype version of ONCOCIN (Shortliffe *et al.*, 1981), has facilitated our analysis of the application area at the knowledge level. As a result, we have developed a domain model that is used in OPAL to instantiate the concepts present in particular cancer protocols. Knowledge of what notions can be anticipated from protocol to protocol guides all of the program's interactions with its user.

Although at the knowledge level there is a single, unified model for cancer therapy, OPAL

must encode the domain model using two different, isomorphic representations. First, OPAL must store entered protocol knowledge in an intermediate representation that can later be translated into a knowledge base for ONCOCIN. The specification of the format for this intermediate representation is one embodiment of the domain model. The second incarnation of the same model is reflected in OPAL's graphical user interface.

OPAL uses two parallel representations of the domain model for logistical reasons. Because of the internal complexity of the graphical representation of the knowledge, protocol descriptions stored within the OPAL interface cannot be transformed directly into knowledge bases for ONCOCIN in a convenient fashion. At the same time, transmuting ONCOCIN production rules, which are "symbol-level" entities (Newell, 1982), back into their corresponding knowledge-level assertions for display to the user is also computationally cumbersome. Thus, the intermediate knowledge representation serves as a useful buffer between the OPAL interface and the target expert system.

We now consider OPAL's domain model in terms of its knowledge-level components and physical implementations. Ontological analysis, a methodology for classifying knowledge-level concepts, has recently been proposed as a general approach for classifying the contents of knowledge bases (Alexander et al., 1986). While we have not attempted to apply ontological analysis to the domain model constructed for OPAL in any rigorous fashion, we will relate our discussion to the taxonomy proposed by Alexander et al. to highlight the correspondence between our respective approaches.

#### **Entities and Relations**

The first component of the domain model consists of an identification of the entities in the application area and their relations. Alexander *et al.* refer to this as the *static ontology*. In the ONCOCIN domain, the "entities" are the therapeutic elements that make up the planning problem (Figure 2). The "relations" are defined in part by the composition hierarchy that specifies how each plan (*e.g.*, treatment according to a protocol) may be defined in terms of more specific plans (*e.g.*, administration of particular chemotherapies). Other relations define attributes of individual entities (*e.g.*, drugs have *dosages* and *routes of administration*).

At the implementation level, the intermediate knowledge representation for a cancer treatment plan is based on a hierarchy of objects describing the necessary *protocol*, *chemotherapy*, *radiotherapy*, and *drug* entities, with appropriate links to establish the composition relations. OPAL creates new objects in this hierarchy and stores and retrieves the values of slots that define attributes of each instance.

The same entity-relation model is implicit in the OPAL user interface. For example, the OPAL form in Figure 1 allows the user to specify the names of the individual drugs that make up a particular chemotherapy. The program presumes that chemotherapies and drugs are entities in the domain and that they are related compositionally. When a user interacts with OPAL, the names of the drugs are entered into the blanks of the graphical form by selecting appropriate choices from a pre-defined menu.

Whenever a blank requesting a drug name is filled in, knowledge stored in the definition of the graphical form causes OPAL to create a new drug object in the intermediate representation and link it to the object for the related chemotherapy. Additional forms in OPAL contain blanks that can define values for the slots in the generated objects. For instance, selecting the box labeled *specify dose* beneath one of the drug names in Figure 1 causes another form to be displayed in which certain properties of the designated drug can be entered. By noting the pathway used to access the new form (*i.e.*, which box in the first form was selected), OPAL can store the knowledge specified via the second form in the appropriate drug object. Thus, the user's path through the various forms in the system determines the particular entities to which entered knowledge is related at each step. Furthermore, whenever knowledge is specified graphically using the OPAL forms (one

embodiment of the domain model), it is simultaneously translated internally to the intermediate representation, whose structure denotes the same model in a different conformation.

#### Domain Actions

Much of ONCOCIN's skeletal planning task is pre-defined by the hierarchical relations among entities in the domain (Figure 2). Initiating a plan to begin a new chemotherapy, for example, tacitly requires invoking plans to administer the component drugs. Because the planning hierarchy is built into OPAL's model of the domain, such actions do not have to be specified explicitly for OPAL when new protocol knowledge is entered. They are assumed constant from protocol to protocol.

On the other hand, there are several ways in which the simple instantiation of the skeletal plans defined by the domain entities can be modified. Fortunately, the list of these possible actions is small. At the knowledge level, we must deal with such concepts as altering the customary dose of a drug, substituting one drug for another, or aborting or delaying chemotherapy. Alexander *et al.* would classify these notions as part of the *dynamic ontology* of the domain.

OPAL's semantic assumptions about the knowledge in ONCOCIN form a "data model" that is derived directly from the domain model. Accordingly, the knowledge-level actions that form the dynamic ontology can be selected directly from menus when physicians enter descriptions of new protocols (Figure 3). The subsequent translation of the intermediate knowledge representation into the knowledge base used by ONCOCIN requires conversion of the knowledge-level specifications into expressions that operate on internal ONCOCIN parameters. The user of OPAL, however, need only be concerned with actions at the knowledge level.

#### Domain Predicates

Just as there is a limited set of actions at the knowledge level, there are limited classes of *conditions* that can cause modification of the basic treatment plan. The results of laboratory tests, certain patient symptoms, and manifestations of treatment-induced toxicity are the principal factors that can dictate particular actions in OPAL's domain.

Much of the knowledge needed to specify ONCOCIN's planning task (the *epistemic* ontology of Alexander et al.) consists of a mapping of specific conditions onto corresponding domain actions. OPAL simplifies knowledge entry by using a different graphical form for each class of conditions. For example, in Figure 4, OPAL displays a predefined list of all the relevant laboratory tests known to the system. When one of the laboratory tests is selected by the user, "rules" predicated on the results of the test can be entered. In Figure 4, the expert has indicated that when a patient's serum creatinine is greater than 1.5, the drug methotrexate should be withheld from "VAM" chemotherapy. The values entered by the physician using OPAL correspond to a production rule: If creatinine is greater than 1.5 then omit the drug. The rule is applicable only within a specific context--in this case, when giving a particular drug (methotrexate) in a particular chemotherapy ("VAM") in a particular protocol. As in the specification of the static attributes of domain entities (Figure 1), the context of the knowledge is determined from the manner in which the user invoked the form.

OPAL must translate predicates based on such knowledge-level concepts as laboratory tests into expressions that evaluate parameters in the ONCOCIN knowledge base. However, the actual production rules required by the inference engine are transparent at the level of the OPAL interface.

## Alterations for Blood Counts



Figure 3: Knowledge-Level Actions in OPAL

Operations that modify the plan of the protocol are specified in OPAL by selecting knowledge-level actions from menus, such as the one toward the right hand side of the figure. These knowledge-level actions are translated into expressions involving ONCOCIN parameters when the knowledge base later is generated from the intermediate representation. The OPAL "form" shown here displays actions to take with the drug methotrexate in a chemotherapy called "VAM". The user is specifying that when a patient's white blood cell count is greater than or equal to 3500 and the platelet count is between 100,000 and 150,000, treatment should be delayed.

#### Procedural Knowledge

Also part of the epistemic ontology is the sequence of chemotherapies and other treatments that must be carried out over time. In OPAL, a special graphical environment is used in which the user can create and link together icons that represent the various procedural elements of oncology treatment plans (Figure 5). Concepts found in traditional programming languages such as sequential control, conditionality, iteration, exception handling, and concurrency can all be specified using a visual syntax that mimics the flowcharts typically found in printed descriptions of protocols. OPAL converts the graphical representation into a finite state table for use by ONCOCIN (Musen *et al.*, 1986c).

# DISCUSSION

Now that tools to build expert systems are readily available commercially, increasing attention is being placed on the problems of creating and maintaining large knowledge bases. In the case of the prototype version of ONCOCIN, developed before work began on OPAL, difficulties in knowledge acquisition were particularly apparent; encoding the system's protocols for lymph node cancer required two years of effort and some 800 hours

TEST: Hematology Granulocytes Hematocrit Hemoglobin Platelets PT PTT WBC Selec	Aikaline Phosphetase Total Bilirubin BUN Creatinine Clearance Scrum Creatinine SGOT SGPT ted Test: Serum C	nistrios	Miscellaneous DLCO ECG Pulm. Function
Granulocytes Hematocrit Hemoglobin Platelets PT PTT WBC Selec	Aikaline Phosphatase Total Bilirubin BUN Creatinine Clearance SGOT SGOT SGPT ted Test: Serum C	:réadhine	Miscellaneous OLCO ECG Pulm. Function
Hematocrit Hemoglobin Platelets PT PTT WBC Selec	Alkaine Prospracese Total Bilirubin BUN Creatinine Clearance Scrum Creatinine SGPT ted Test: Serum C	irèathine Sui	DLCO ECG Pulm. Function
Hemoglobin Platelets PT PTT WBC Selec	BUN Creatinine Clearance Securi Creatinine SGOT SGPT ted Test: Serum C	irèathine Sui	ECG Pulm. Function
Platelets PT PTT WBC Selec	Creatinine Clearance Serum Creatinine SGOT SGPT ted Test: Serum C	réathine Sui	bcycle:
PT PT WBC Salec	SGOT SGPT SGPT Serum Creatining SGPT Serum Creating Serum Creating SGPT Serum Creating SGPT Serum Creating Seru	réatinine Sui	bcycle:
PTT WBC Selec	SGOT SGPT ted Test: Serum C	risatinine Sui	bcycle:
wec Selec	SGPT ted Tast: Serum C	1 :risatinine Sui	bcycle:
wec Selec	sGPT ted Test: Serum ( wotherapy: VAM	reatinine Sui	bcycle:
Selec	ted Test: Serum C	:reatinine 	bcycle:
	otherapy:VAM	Sul	bcycle:
	otherapy:VAM	Sui	bcycle:
	otherapy: <u>VAM</u>	Su	bcycle:
Test Alterations for Chem			
	a		
43846	Action	Value	Action
	· · ·		
Test Alterations for Drug	METHOTREXATE	(You must se	ect a chemotherapy first)
Vaka	Action	Votum	Action
Y LIUS	Acubri	V 2/08	ACIDI
> 1.5	Withhold		
		-	

#### Figure 4: Knowledge-Level Rules in OPAL

The blanks in this form allow the expert to specify how the results of laboratory tests should cause modification of the protocol. The knowledge entered is automatically converted to production rules that can be invoked by the ONCOCIN inference engine. In this example, the user has specified that when a the patient's serum creatinine is greater than 1.5, the drug methotrexate should be withheld from "VAM" chemotherapy.

of an oncology expert's time. The addition of three protocols for breast cancer chemotherapy took several more months, most of which involved the usual cycles of programming by knowledge engineers and system testing by domain experts.

A principal goal of our current work is to solve the practical problem of expediting knowledge acquisition for ONCOCIN by reducing dependence on knowledge engineers as intermediaries. In OPAL, we have attempted to ease the interchange between experts and the computer by providing a visual model of oncology knowledge that is familiar to physicians. The model incorporates sufficient assumptions about the domain that clinicians can describe new protocols by means of a simple "fill in the blanks" approach.

Other workers have emphasized the usefulness of visual representations in facilitating knowledge entry by experts. ETS (Boose, 1985), for example, allows construction of knowledge bases for certain classification tasks. Once a prototype knowledge base has been specified by means of a textual dialog, ETS displays a directed graph that allows the user to visualize how classifying particular entities in the application area may entail additional classifications. Another program called ESSA (Hannan and Politakis, 1985), on the other hand, allows users themselves to define classification hierarchies graphically. The actual knowledge, however, must be specified as production rules in the EXPERT language, thus requiring that the user be familiar with a particular expert system "shell". Neither ETS nor ESSA uses graphics for specification or display of knowledge-level concepts.

## Alterations for Blood Counts

Drug Combination:		VAM	Subcycle;	<u></u>
Drug:	METHOTREXATE			
			Attenuate Dose Withhold Drug Substitute Drug Consult Delay	
WBC (x 1000)	>= 150	Platelets (× 1000) 100 - 150	Abort Report New protocol Off protocol Display Skip cycle Order Test	< 75
>= 3.5 1	00% of STD		CLEAR	
3.0 - 3.5				
2.5 - 3.0				
< 2.5		·		

Figure 3: Knowledge-Level Actions in OPAL

Operations that modify the plan of the protocol are specified in OPAL by selecting knowledge-level actions from menus, such as the one toward the right hand side of the figure. These knowledge-level actions are translated into expressions involving ONCOCIN parameters when the knowledge base later is generated from the intermediate representation. The OPAL "form" shown here displays actions to take with the drug methotrexate in a chemotherapy called "VAM". The user is specifying that when a patient's white blood cell count is greater than or equal to 3500 and the platelet count is between 100,000 and 150,000, treatment should be delayed.

#### Procedural Knowledge

Also part of the epistemic ontology is the sequence of chemotherapies and other treatments that must be carried out over time. In OPAL, a special graphical environment is used in which the user can create and link together icons that represent the various procedural elements of oncology treatment plans (Figure 5). Concepts found in traditional programming languages such as sequential control, conditionality, iteration, exception handling, and concurrency can all be specified using a visual syntax that mimics the flowcharts typically found in printed descriptions of protocols. OPAL converts the graphical representation into a finite state table for use by ONCOCIN (Musen *et al.*, 1986c).

# DISCUSSION

Now that tools to build expert systems are readily available commercially, increasing attention is being placed on the problems of creating and maintaining large knowledge bases. In the case of the prototype version of ONCOCIN, developed before work began on OPAL, difficulties in knowledge acquisition were particularly apparent; encoding the system's protocols for lymph node cancer required two years of effort and some 800 hours

	Atterations	for Lab Tests	
TEST:			
Hematology	Chem	Miscellaneous	
Grenulocytes	Alkaline Phosphetase		DLCO
Hematocrit	Total Billirubin		ECG
Hemoglobin	BUN		Puim. Function
Platelets	Creatinine Clearance		
PT ·	Serum Creatinine		
PTT	SGOT	t	
WBC	SGPT		
<b>C</b> -1	and Tanda Samura C	ninetiala.	
Select	ed iest: Serum C	reaunine	
Test Alterations for Chem	VAM	Subcycle:	
	unn apy		
Value	Action	Value	Action
and a straight and a straight and a straight		·	
( <u></u>			
Test Alterations for Drugt	METHOTREXATE	You must select a che	motherany first)
		_ (	
Value	Action	Value	Action
> 1.5	Withhold		
		••	
<u></u>			
			•
	· · · · · · · · · · · · · · · · · · ·	o e	
	· · · · · · · · · · · · · · · · · · ·		

## Figure 4: Knowledge-Level Rules in OPAL

The blanks in this form allow the expert to specify how the results of laboratory tests should cause modification of the protocol. The knowledge entered is automatically converted to production rules that can be invoked by the ONCOCIN inference engine. In this example, the user has specified that when a the patient's serum creatinine is greater than 1.5, the drug methotrexate should be withheld from "VAM" chemotherapy.

of an oncology expert's time. The addition of three protocols for breast cancer chemotherapy took several more months, most of which involved the usual cycles of programming by knowledge engineers and system testing by domain experts.

A principal goal of our current work is to solve the practical problem of expediting knowledge acquisition for ONCOCIN by reducing dependence on knowledge engineers as intermediaries. In OPAL, we have attempted to ease the interchange between experts and the computer by providing a visual model of oncology knowledge that is familiar to physicians. The model incorporates sufficient assumptions about the domain that clinicians can describe new protocols by means of a simple "fill in the blanks" approach.

Other workers have emphasized the usefulness of visual representations in facilitating knowledge entry by experts. ETS (Boose, 1985), for example, allows construction of knowledge bases for certain classification tasks. Once a prototype knowledge base has been specified by means of a textual dialog, ETS displays a directed graph that allows the user to visualize how classifying particular entities in the application area may entail additional classifications. Another program called ESSA (Hannan and Politakis, 1985), on the other hand, allows users themselves to define classification hierarchies graphically. The actual knowledge, however, must be specified as production rules in the EXPERT language, thus requiring that the user be familiar with a particular expert system "shell". Neither ETS nor ESSA uses graphics for specification or display of knowledge-level concepts.



Figure 5: Procedural Knowledge in OPAL

This is a modified version of the sequence of steps in a protocol for lung cancer entered into OPAL. Patients are randomly assigned to either Arm A, which tests the chemotherapies "VAM" and "POCC", or Arm B, which tests the chemotherapy "CAVP". Patients who respond to either treatment arm may be randomly assigned to receive "PCI" radiation treatments. The visual flowchart in OPAL is converted to a finite state table for use by ONCOCIN.

Knowledge entry tools such as INKA (Phillips et al., 1985) come closer to using visual representations in the manner adopted by OPAL. INKA permits experts to employ a structured subset of natural language to define PROLOG rules for diagnosis of faults in electronic circuits. The program displays a schematic diagram of the circuit under discussion, specified in advance by a knowledge engineer. Experts simply point to components in the diagram when composing new rules, causing descriptions of the indicated circuit elements to be incorporated within the entered text. INKA relies on this graphical display of domain objects to acquire part of the knowledge from experts. The program also incorporates knowledge-level concepts such as "transistor" and "voltage" into its structured rule language. INKA thus adopts a strong domain model that facilitates knowledge entry. However, the domain model in INKA is *not* the data model. The program's data model is one in which the knowledge base is viewed as a collection of "rules"; there is no framework that can help the domain expert recognize what knowledge is *expected* by the system. As a result, INKA itself cannot inform the expert if any rules are missing or whether a new rule interacts with previously entered statements.

Because the data model in OPAL directly reflects the domain model, physicians learn immediately that defining protocol knowledge for ONCOCIN is simply a matter of filling out all the OPAL "forms". Once a particular fact has been entered into OPAL, the appropriate blanks remain filled in unless the knowledge is later retracted by the expert; previously entered knowledge can therefore be perused easily at any time. Because entering new knowledge into a blank overwrites prior specifications, contradictions in the knowledge base can be avoided. At the same time, portions of the knowledge can be recognized as missing if particular blanks are not filled in.

Previous knowledge editors have been based on either of two types of data models. The first and most common approach has been to provide a model of *the knowledge* 

representation itself. Typical expert system building shells such as EMYCIN and more specialized tools such as INKA have adopted this type of data model. TEIRESIAS (Davis, 1979), for example, allowed its user to examine and modify faulty knowledge in the MYCIN system. TEIRESIAS was a landmark program in that it demonstrated that inferring a domain model from an existing knowledge base could be used to assure the syntactic and semantic integrity of new knowledge entered by an expert. However, the data model of the MYCIN knowledge base presented by TEIRESIAS to the user-that of a collection of rules--was the same model adopted by the consultation program. Although the system took great pains to attempt translation between natural language and LISP code, if an expert could not convey his or her thoughts in terms of MYCIN-style rules, it was impossible to interact with TEIRESIAS.

The second, more abstract approach has been to represent knowledge using a model of a *problem-solving strategy*. ROGET (Bennett, 1985), for example, acquires knowledge for EMYCIN-based systems using a model akin to heuristic classification (Clancey, 1985). By asking the user what hypotheses are to be considered and what kinds of evidence can support or refute those hypotheses, the EMYCIN rule-base for a new expert system can be produced from what is essentially a knowledge-level analysis of the domain. In a similar fashion, ETS (Boose, 1985) can generate prototype knowledge bases using a weaker model of classification based on "personal construct theory", a system used by psychometrists for evaluating how people categorize elements in their environment. Because ETS and ROGET are derived from such general strategic models, the systems are most useful in eliciting the basic conceptual structure of a knowledge base; neither is intended to generate production-version expert systems.

A system called SALT (Marcus *et al.*, 1985) models a different problem-solving strategy, one for *constructing solutions*. The program is reported to have actually been used by engineering experts to specify knowledge for configuring elevators in new buildings. SALT requires a structured language for entry of specific concepts such as methods for determining values, constraints on values, and corrections for constraint violations. The system then generates rules in OPS5. Like ROGET, SALT assumes that the user will be able to conceptualize the solution to the problem in terms of the model provided by the system. Although it may be easier for a non-programmer to think about *strategies* rather than *rules*, the approach may still require the expert to enter knowledge in an unfamiliar format.

The data model used in OPAL, based on the domain model itself, is perhaps the most categorical way in which the contents of a knowledge base can be viewed. Unlike previous knowledge acquisition tools, OPAL's data model is simply one of what knowledge should be expected. As a result, the user is not given the flexibility found in other systems to specify new concepts. Novel instantiations of existing concepts can be entered (*e.g.*, a previously unknown drug can be defined), but the general classes of concepts in OPAL are predetermined. The domain model, however, is generally sufficient because of the highly structured, stylized nature of oncology treatment plans.

The limitations in OPAL are the limitations of the domain model. OPAL could obviate the need for knowledge engineers entirely if the domain model were complete enough. Yet regardless of the thoroughness with which one can understand the application area at the knowledge level, it is impossible to anticipate all of the constructs one might encounter in oncology protocols. Even if one had the necessary prescience, designing acceptable graphical forms to capture the knowledge for such an all-inclusive model would be unwieldy. Knowledge acquisition is nevertheless expedited when physicians can use OPAL for the bulk of the knowledge entry, recognizing that specialized knowledge editing by computer scientists may often be necessary later. Previous workers have stressed the great advantages of having preliminary knowledge specification performed by domain experts working alone (Boose, 1985; Phillips *et al.*, 1985). Our goal in OPAL is to maximize the knowledge that experts can enter independently by providing a data model that approximates the way oncologists seem to think about the application area.

OPAL's strategy of providing a model of the domain knowledge to guide the acquisition of new protocols for ONCOCIN is possible because of the structured nature of cancer treatment plans. The approach is practical in our setting primarily because encoding oncology protocols is a repetitive task that can be greatly accelerated when physicians enter the knowledge themselves. Developing expert systems in other application areas, such as process control and fault diagnosis for electronic instruments (Phillips *et al.*, 1985), similarly requires the reiterative specification of knowledge bases that are in many ways homologous. Tools that allow experts to define new, congenerous knowledge bases in terms of explicit, knowledge-level models should greatly facilitate creation of expert systems in these domains.

## ACKNOWLEDGEMENTS

This work has been supported by grants LM-04420, LM-07033, RR-01631, and LM-04316 from the National Institutes of Health. Computer facilities were provided by the SUMEX-AIM resource under NIH grant RR-00785 and through gifts from Xerox Corporation and Corning Medical. Dr. Musen and Dr. Shortliffe have also received support from the Henry J. Kaiser Family Foundation. Joan Walton implemented the first prototype version of OPAL. We thank oncologists Joel Bernstein, Robert Carlson, Charlotte Jacobs, and Richard Lenon for their evaluations of the developing system. We are grateful to Michael Kahn, Samson Tu, and Cliff Wulfman for valuable discussions.

# REFERENCES

- Alexander, J.H., Freiling, M.J., Shulman, S.J., Staley, J.L., Rehfuss, S., and Messick, S.L. Knowledge level engineering: Ontological analysis, in Proceedings AAAI-86, pages 963-968, American Association for Artificial Intelligence, Philadephia, PA, August, 1986.
- Bennett, J.S. ROGET: A knowledge-based system for acquiring the conceptual structure of a diagnostic expert system. Journal of Automated Reasoning, 1985, 1(1), 49-74.
- Boose, J.H. A knowledge acquisition program for expert systems based on personal construct psychology. International Journal of Man-Machine Studies, 1985, 23, 495-525.
- Clancey, W.J. Heuristic Classification. Artificial Intelligence, 1985, 27, 289-350.
- Davis, R. Interactive transfer of expertise: Acquisition of new inference rules. Artificial Intelligence, August 1979, 12(2), 121-157.
- de Greef, P. and Breuker, J. A case study in structured knowledge acquisition, in Proceedings of the Ninth International Joint Conference on Artificial Intelligence, pages 390-392, Los Angeles, California, August, 1985.
- Fox, J., Myers, C.D., Greaves, M.F., and Pegram, S. Knowledge acquisition for expert systems: Experience in leukemia diagnosis. *Methods of Information in Medicine*, April 1985, 24(2), 65-72.
- Friedland, P.E. and Iwasaki, Y. The concept and implementation of skeletal plans. Journal of Automated Reasoning, 1985, 1(2), 161-208.
- Hannan, J.J. and Politakis, P. ESSA: An approach to acquiring decision rules for diagnostic expert systems, in The Second Conference on Artificial Intelligence Applications, pages 520-525, IEEE Computer Society Press, Miami, FL, December,

1985.

- Lane, C.D. Ozone Reference Manual. Technical Report KSL-86-40, Knowledge Systems Laboratory, Stanford University, 1986.
- Marcus, S., McDermott, J., and Wang, T. Knowledge acquisition for constructive systems, in Proceedings of the Ninth International Joint Conference on Artificial Intelligence, pages 637-639, August, 1985.
- Musen, M.A., Combs, D.M., Walton, J.D., Shortliffe, E.H., and Fagan, L.M. OPAL: Toward the computer-aided design of oncology advice systems, in Proceedings of the Tenth Annual Symposium on Computer Applications in Medical Care, Washington, DC, October, 1986.
- Musen, M.A., Rohn, J.A., Fagan, L.M., and Shortliffe, E.H. Knowledge engineering for a clinical trial advice system: Uncovering errors in protocol specification, in Proceedings of AAMSI Congress 86, pages 24-27, American Association for Medical Systems and Informatics, Anaheim, California, May, 1986.
- Musen, M.A., Fagan, L.M., and Shortliffe, E.H. Graphical specification of procedural knowledge for an expert system, in Proceedings of the 1986 IEEE Computer Society Workshop on Visual Languages, pages 167-178, Dallas, TX, June, 1986.

Newell, A. The knowledge level. Artificial Intelligence, 1982, 18, 87-127.

- Phillips, B., Messick, S.L., Freiling, M.J., and Alexander, J.H. INKA: The INGLISH knowledge acquisition interface for electronic instrument troubleshooting systems, in The Second Conference on Artificial Intelligence Applications, pages 676-681, Miami, FL, December, 1985.
- Shortliffe, E.H., Scott, A.C., Bischoff, M.B., van Melle, W., and Jacobs, C.D. ONCOCIN: An expert system for oncology protocol management, in Proceedings of the Seventh International Joint Conference on Artificial Intelligence, pages 876-881, Vancouver, B.C., August, 1981.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# Explanation-Based Learning for Knowledge-Based Systems

Michael J. Pazzani The Aerospace Corporation P.O.Box 92957 Los Angeles, CA 90009 and UCLA Artificial Intelligence Laboratory 3531 Boelter Hall Los Angeles, CA 90024

## Abstract

We discuss explanation-based learning for knowledge-based systems. First, we identify some potential problems with the typical means of acquiring a knowledge base: interviewing domain experts. Next, we review some examples of knowledge-based systems which include explanation-based learning modules and discuss two systems in detail: ACES (Pazzani, 1986a) which learns heuristics for fault diagnosis from device descriptions, and OCCAM (Pazzani, 1986b) which learns to predict the outcome of economic sanction episodes from simple economic theories. We conclude that explanation-based learning is a promising approach to constructing knowledge-based systems when the required information is available but not in the form of heuristic rules. In this case, the role of explanation-based learning is to explicate heuristics which are only implicit in deep models.

## Introduction

The typical image of the development of a knowledge-based system is that a domain expert is interviewed to explain how he solves a problem in a given area. In the interview, the expert reveals heuristics which can be encoded as rules in a knowledge-based system. For some applications, there are a number of potential difficulties with the development of knowledge-based in this manner. Many of these difficulties could be avoided if the rules for a knowledge-based system could be learned. Potential problems which must be addressed in the development of knowledge-based systems include:

- Availability of Experts. Domains which are selected for development of knowledge-based systems are often those domains with few experts. By definition, an expert is someone who performs extremely well in a problem area. It may be more important for the expert to continue to solve problems in his area than to devote time to the development of a knowledge-based system.
- Existence of Experts. In some areas, there may not exist experts with enough experience to possess the heuristics required by knowledge-based systems. Consider how knowledge-based systems in medicine are developed: typically a physician will report on the symptoms of various diseases and possibly the certainty of having a disease given a set of symptoms. This empirical association between symptoms and diseases summarizes knowledge gained through years of experience. Now, consider if the same methodology could be applied to a system to diagnose faults in a new satellite. The experts in this case are not really experienced in diagnosis of this particular satellite. The expert most probably will be

someone who is experienced with diagnose of a different satellite, or a designer of the satellite. It will be a difficult task for either type of expert to give empirical associations between atypical behavior and faulty components. This difficulty will be present in any new system. The problem is especially serious if only a few systems are manufactured. There may never be enough failures to develop a human expert to "knowledge engineer".

• Experts' ability to report on the processes used in making a decision. A critical assumption in developing a knowledge-based system by interviewing an expert is that an expert is able to reveal the heuristics he uses to make decisions. In some areas, such as vision and natural language understanding, people do not even attempt to explain their decisions. For example, consider the sentence "Many words have more than one sence". How did you know the word "sence" is "sense" misspelled and means "meaning" rather than "capacity to perceive stimuli" or any other meaning? For those who answer the previous question: were you consciously aware of this when making the decision or are you really answering the question "How could one (rather than how did I) make the decision"?

Even when an expert is willing to give an explanation for making a decision, can he really tell us how he made the decision? When a stock broker recommends investing in United Hotcakes Preferred does he have conscious access to the processes he used to come to the conclusion that this stock is a good investment? The psychological evidence is mixed. For example, (Berl et al., 1976) students verbal reports of their decision criteria for deciding on a college to attend matched well with their apparent actual criteria. However a number of discrepancies have been found between the information reportedly used to influence a decision and the information actually used.

- Failure to report an influential factor. For example, in one experiment (Nisbett and Wilson, 1977), subjects were given electric shocks of increasing intensity. Some subjects were given a placebo pill which they were told would produce hand tremors, butterflies in the stomach (i.e., the symptoms of electric shock). This experiment was performed to demonstrate that those subjects who attributed the symptoms of the shock to the pill would tolerate more shock than those who did not. In fact, the subjects who received the placebo pill withstood an average of four times the amperage of those who did not. However, 75% of the subjects who received the placebo pill and tolerated an extreme amount of shock did not report the pill as the explanation.
- Erroneous reporting of noninfluential factors. In one experiment (Nisbett and Ross, 1978), some subjects viewed a film while a power saw made a distracting noise and the control subjects viewed the film with no distraction. Subjects then rated the film according to how interesting they thought it to be, how much they thought others would be affected by it and how sympathetic they found the main character to be. Although there was no detectable difference of the rating between the two groups, 55% of the subjects who were distracted by the power saw felt that the distraction had lowered their ratings.

Ericson and Simon (Ericsson and Simon, 1984) have proposed a model which accounts for some of the conditions which lead to excluding relevant information from protocols as well as including irrelevant information. At best, however, information arrived at by interviewing domain experts is only the tip of the iceberg. At worst, this information can be misleading when the expert is actually answering the questions "How might one solve this sort of problem?" rather than "How did I solve this problem?".

 Changes in the underlying theory require changes in the knowledge-base. Consider the problem of several companies which built financial knowledge-based systems. After spending a considerable amount of time and effort interviewing tax experts, the tax laws have changed. More interviews with tax experts are now needed to encode the implications of the new tax code as heuristics which indicate financial strategies. In general, when changes in the underlying theory occur, the knowledge acquisition process may need to be repeated to encode changes as new heuristics.

One way to avoid the above potential problems is to eliminate the knowledge engineering's task of interviewing an expert by having the machine learn its own heuristics. We are exploring this issue in a number of contexts: ACES (Pazzani, 1986a) which identifies faults in the attitude control system of the DSCS-III satellite<sup>1</sup> and OCCAM (Pazzani, 1986b) which predicts the outcome of economic sanction episodes. Both of these programs use explanation-based learning techniques. For example, in ACES, rather than <u>inducing</u> diagnosis heuristics (i.e., empirical associations between symptoms and device failures) from a number of training examples (Michie, 1983), diagnosis heuristics are <u>deduced</u> as needed from device models which describe the system functionality and connectivity.

The idea of explanation-based learning of heuristic rules fits in well with Steels' notion of a Second Generation Expert Systems (Steels, 1986). For a variety of reasons, including enhanced explanation capabilities, and robustness, Steels argues that expert systems should contain both "shallow" associations (i.e., heuristics) which allow for rapid solutions to commonly encountered problems and "deep" causal models which enable the system to solve uncommon or even unanticipated problems with general search techniques. An important advantage of second generation expert systems lies in the area of knowledge acquisition. After searching for a solution to a problem in the deep model, a heuristic can be created. Explanation-based learning is one such technique for creating heuristics.

# **Explanation-Based Learning**

Let us briefly examine the problem of how explanation-based learning might be used to create heuristics for financial expert systems. In the proposed new tax law, the interest expense on a home mortgage is tax deductible, but the interest on an automobile loan is not. A suitable representation of the tax law would serve as a deep model. Let us assume that the problem that the financial expert system is intended to solve is to maximize net income. One aspect of maximizing net income is minimizing income taxes. A general search program, with operators such as take out an automobile loan, and take out a second mortgage, would find that the taking out a second mortgage on one's home to pay for an automobile would result in lower income taxes. After, this expensive general search is done once, the results of the search can be saved as a heuristic:

IF the goal is to minimize taxes, AND the client wants to buy an automobile THEN take out a second mortgage to pay for the car

In addition, the conditions which are necessary to perform the operator (take out a second loan), can be propagated to the heuristic as additional preconditions (Minton, 1984):

IF the goal is to minimize taxes, AND the client wants to buy an automobile AND the client owns a home AND the equity in the home is greater than price of the car THEN take out a second mortgage to pay for the car

Explanation-based learning systems (DeJong, 1983, Mitchell et al., 1986a) share a common approach to generalization. First, an example problem is solved producing an explanation (occasionally called a justification, or a proof) which indicates what information (e.g., features of the example and inference rules) was needed to arrive at a solution. Next, the example is generalized by retaining only those features of the example which were necessary to produce the explanation. Various systems differ according to the problem solved, and who does the problem solving. For example, in LEAP (Mitchell et al., 1986b) a user designs a VLSI circuit to achieve some specified functionality. LEAP produces a justification which indicates how the circuit implements the specified function. In OCCAM, the reason that an economic sanctions incident failed or succeeded to achieve the desired effect determines which features of the incident should be generalized. In ACES, an explanation is produced which indicates why a fault proposed by a fault diagnosis heuristics was not confirmed by device models.

<sup>&</sup>lt;sup>1</sup>The attitude control system is responsible for detecting and correcting deviations from the desired orientation of the satellite.

# Fault Diagnosis

Two different approaches have been used for fault diagnosis. In one approach (Davis, 1982, Genesereth, 1981, Scarl, 1985), the observed functionality of devices are compared to their predicted functionality which is specified by a quantitative or qualitative model of the device (de Kleer and Brown, 1984, Kuipers, 1984, Forbus, 1984). For a large system, such as a satellite, with a number of rapidly changing data values, comparing observed to predicted functionality can be inefficient. The alternative approach (Shortliffe, 1976, Nelson, 1982, Wagner, 1983) encodes empirical associations between unusual behavior and faulty components as heuristic rules. This approach requires extensive debugging of the knowledge base to identify the precise conditions which indicate the presence of a particular fault. In previous work, (Pazzani and Brindle, 1985, Pazzani and Brindle, 1986) we have described the Attitude Control Expert System (ACES) in which these two approaches are integrated. Heuristics examine the atypical features and hypothesize potential faults. Device models confirm or deny hypothesized faults. Thus, heuristics focus diagnosis by determining which device in a large system might be at fault. Device models determine if that device is indeed responsible for the atypical features.

Here, we address the problem of revising the fault diagnosis heuristics when they hypothesize a fault which is later denied. This occurs when all of the possible exceptions to a heuristic are not explicitly stated. When a fault is proposed, and later denied by device models, the heuristic which suggested the fault is revised so that the hypothesis will not be proposed in future similar cases.

ACES learns how to avoid a hypothesis failure after just one example. It does this by finding the most general reason for the hypothesis failure. Device models provide an explanation for the hypothesis failure. The device models indicate which features would have been needed to be present (or absent) to confirm the hypothesis. Explanation-based learning improves the performance of ACES by creating fault diagnosis heuristics from information implicit in the device models.

## Explanation-based Learning of Fault Diagnosis Heuristics

The basic idea behind the learning of fault diagnosis heuristics is that simulating a fault with device models will result in a number of predictions. If these predictions are not present in a particular system, the fault which was simulated can be ruled out. When a hypothesized fault is not confirmed, in addition to ruling out the fault in the current case, the heuristic which suggested the fault can be revised to not propose the fault in future similar cases. The top-level diagnosis algorithm is presented in Figure 1.<sup>2</sup>

To diagnose a fault in circuit, First, suggest a fault with diagnosis heuristics, Then confirm (or deny) the fault with device models.

```
diagnose(Circuit,Fault) :-
    problem(Fault,Rule_id),
    confirm(Circuit,Fault,Rule_id).
```

Figure 1: Top level diagnosis algorithm

A simple example of explanation-based learning of diagnosis heuristics should help the presentation of the learning algorithm. Consider the circuit in Figure 2. The circuit consists of two light bulbs (a parking light and a taillight) in parallel protected by a fuse.

<sup>&</sup>lt;sup>2</sup>This version of ACES is implemented entirely in PROLOG. By providing English descriptions in addition to PROLOG rules, I hope that anyone will be able to read this section, and that there is enough detail for those who wish to experiment with ACES.



Figure 2: A simple circuit consisting of two light bulbs in parallel protected by a fuse.

ACES utilizes three types of data:

1. Diagnosis Heuristics: shallow associations between atypical data values and faults. Two simple diagnosis heuristics which define a blown fuse and a burnt out light bulb are illustrated in Figure 3. These diagnosis heuristics are interpreted by a PROLOG meta-interpreter. In the second rule, the term,

device\_functionality(L,\_Any\_input, not\_working)

gives the modified qualitative functionality for a burnt out light bulb: Given any input, the light bulb does not work. The normal functionality for a light bulb,

device\_functionality(L,\_Any\_input,\_Any\_input)

indicates that if the input to a light is working, then the light bulb is working and if the input is not working, the light bulb will not work.

- 2. Device Models: a description of the connectivity of a circuit and the functionality of the components. Figure 4 illustrates the definition for the circuit in Figure 2.
- 3. Measurements: Observed data from the device. For this example, let us assume that the battery is working, the taillight is working and the parking light is not working (see Figure 5).

```
IF a device is not working
THEN the fuse of the device's circuit is blown.
symptom(device_functionality(F,_Any_input,not_working),rule_001) <=
      value(D,not_working) &
      isa(D,device) &
      connected(F,D) &
      isa(F,fuse).
IF a light is not working
THEN the light is burnt out.
symptom(device_functionality(L,_Any_input,not_working),rule_002) <=
      value(L,not_working) &
      isa(L,light).
Figure 3: Initial diagnosis heuristics.
```

```
circuit(c1,[battery_1,fuse_1,taillight,parking_light,ground_1]).
connected(battery_1,fuse_1).
connected(fuse_1,taillight).
connected(fuse_1,parking_light).
connected(taillight,ground_1).
connected(parking_light,ground_1).
isa(battery_1,battery).
isa(ground_1,ground).
isa(fuse_1,fuse).
isa(fuse_1,fuse).
isa(taillight,light).
isa(parking_light,light).
isa(light,device).
isa(fuse,device).
device_functionality(fuse,Input,Input).
```

Figure 4: Description of functionality and connectivity for the circuit in Figure 2.

```
value(taillight, working).
value(parking_light, not_working).
value(battery_1, working).
```

Figure 5: Initial data.

When diagnosis starts the first rule in Figure 3 (rule\_001) will suggest that the fault is a blown fuse (fuse\_1) since a device (parking\_light) protected by the fuse is not working. The next step is to confirm or deny the fault using device models. A qualitative simulation of the circuit with the fuse blown yields a prediction that the taillight will also not be working. This prediction is denied since the taillight is in fact working. Therefore, the hypothesis that the fuse is blown can be ruled out. In addition, the rule which proposed the hypothesis can be revised so that the hypothesis failure does not occur in future similar cases. (After rule\_001 is revised, rule\_002 suggests that the bulb of the parking light is burned out. This fault is confirmed by the device models.) To illustrate how this learning occurs and what exactly is meant by "similar cases", it is necessary to describe the confirmation process in more detail (see Figure 6).

When the qualitative simulation generates a prediction, it also returns a justification of how the prediction was arrived at. This justification consists of the conditions which were needed to establish the prediction. In the current example, the justification consists of the fact that the the taillight is a light, and the taillight is connected to the fuse (see Figure 7). The justification indicates that no specific knowledge about the taillight or fuse\_1 was needed to make the prediction. The same prediction would be made for any light which was connected to any blown fuse.

When a prediction fails, the learning process is initiated (Schank, 1982). The fault diagnosis heuristic which proposed the fault is modified to include a test for the violated prediction in all future cases which would generate the same prediction for the same reason (i.e, have the same justification). To continue our example, rule\_001 will be modified so that in future cases, to propose a blown fuse, if there is a light connected to the fuse, the light must not be working. The revised version of rule\_001 is shown in Figure 8.

```
To confirm a fault:
First generate a prediction,
Next see if the prediction is false
If the prediction is false,
        revise the rule which generated the prediction
        and rule out the fault
         (otherwise, try to generate a new prediction)
Succeed when there are no more predictions are generated.
confirm(Circuit, Fault, Rule_id) :-
        prediction (Circuit, Fault, Prediction, Justification),
        denied (Prediction)
        fix_rule_failure(Rule_id,Fault,Prediction,Justification)
         !,
        fail.
confirm(_,_,_).
         Figure 6: Confirmation: a failed prediction initiates the learning process.
fault:
                 device_functionality(fuse_1,_Any_input, not_working)
prediction:
                 value(taillight, not_working)
from database: value(taillight, working)
justification: connected(fuse 1, taillight) & isa(taillight, light)
           Figure 7: Violated prediction of a blown fuse and it's justification.
symptom(device_functionality(F,_Any_input,not_working),rule_001) <=</pre>
        value (D, not_working) &
        isa(D, device) &
        connected (F, D) &
        isa(F,fuse)
        if connected (F, V037) & isa (V037, light) & F \== V037
            then value (V037, not working)
            <u>else true.</u>
```

Figure 8: Revised heuristic which proposes a blown fuse- changes <u>underlined</u>.

## A Definition of Learning Diagnosis Heuristics

More formally, a diagnosis heuristic can viewed as the implication:

```
F and consistent(H) \rightarrow H
```

where **F** is a set of features, **H** is a hypothesis, and **consistent(H)** is true if believing **H** does not result in a contradiction.<sup>3</sup> In the previous example, **H** corresponds to:

device\_functionality(fuse\_1,\_Any\_input, not\_working)

and F corresponds to:

```
value(D,not_working) &
isa(D,device) &
connected(F,D) &
isa(F,fuse).
```

In our approach to learning and fault diagnosis, consistent(H) is the hypothesis confirmation process which makes use of information in device models. The confirmation process can be viewed as the following implications:

 $H \rightarrow M_{H}$ 

```
J_1 \text{ and } M_H \rightarrow P_1
J_i \text{ and } M_H \rightarrow P_i
J_n \text{ and } M_H \rightarrow P_n
```

where  $M_H$  is a new device model which is the same as the correct (or current) model except that one device (the broken one) has a different functionality. This new device model has a number of predictions  $P_i$  which are checked for consistency with the observed functionality of the device. In general, the predictions  $P_i$  of a modified device model  $M_H$  only hold under certain conditions  $J_i$ .

If in a particular example, F is true, but consistent(H) is false because not(P<sub>i</sub>) is true, then the diagnosis heuristic can revised to<sup>4</sup>:

F and J<sub>i</sub> and P<sub>i</sub> and consistent(H)  $\rightarrow$  H F and not(J<sub>i</sub>) and consistent(H)  $\rightarrow$  H

In the previous example J<sub>i</sub> is:

connected(F,V037) & isa(V037,light)

and P<sub>i</sub> is:

value (V037, not\_working).

The point of revising diagnosis heuristics is that it is less expensive computationally to rule out a hypothesis by testing for  $P_i$  than proving consistent(H). In the above example, after rule\_001 is modified, only one fault hypothesis rather than two are required to correctly identify the failure as a blown light bulb for the parking light. The number of logical inferences required drops from 451 to 305.

<sup>&</sup>lt;sup>3</sup>See (Charniak et al., 1980), for a more thorough discussion of "consistent".

<sup>&</sup>lt;sup>4</sup>We use the equivalent if-then-else construct to avoid proving J<sub>1</sub> again.

## Learning Heuristics to Diagnosis Attitude Control Anomalies

Explanation-based learning has proved useful in satellite diagnosis. Figure 9 presents the definition of two fault diagnosis rules. The rules in this version of ACES have a LISP-like syntax since the satellite diagnosis system is written in a version of PROLOG implemented in LISP. The first element of a list is the predicate name. Variables are preceded by "?". The part of the rule preceded by ":-" is a fault hypothesis, and the part of the rule after ":-" is the conditions which are necessary to be proved to propose the hypothesis. These rules implement two very crude diagnosis heuristics: "if the speed of a reaction wheel is 0, then the tachometer is broken" and "if the speed of a reaction wheel is 0, then the

Figure 9: Initial Fault Diagnosis Heuristics.

Some information about the attitude control system should help to follow the example. The attitude control system consists of a number of sensors which indicate the attitude (i.e., the orientation) of the satellite by the position of the sun and the earth. In addition, there are a set of four reaction wheels whose speed can be varied to adjust the momentum (and therefore the attitude) of the satellite. The speed of the reaction wheels as measured by a number of tachometers is also used as an estimate of attitude to provide finer control of the attitude.

In this example, five atypical features are detected:

- 1. (value-violation WSPR+ 9:40 15:00 0): The value of the pitch-roll+ wheel speed is 0 from 9:40 until the end of the sample period at 15:00.
- 2. (rate-violation WSPR+ 8:00 9:40 -200 0): The value of the pitch-roll+ wheel speed changed an atypical amount (from -200 to 0) between 8:00 and 9:40.
- 3. (rate-violation WSPR- 8:00 9:40 -300 -100): The value of the pitch-roll+ wheel speed changed an atypical amount (from -300 to 100) between 8:00 and 9:40.
- 4. (rate-violation WSPY+ 8:00 9:40 -200 -400): The value of the pitch-yaw+ wheel speed changed an atypical amount (from -200 to -400) between 8:00 and 9:40.
- 5. (rate-violation WSPY- 8:00 9:40 -100 -300): The value of the pitch-yaw- wheel speed changed an atypical amount (from -100 to -300) between 8:00 and 9:40.

The first rule in Figure 9 suggests that the cause of the value violation is a broken tachometer of the pitch-yaw+ wheel. Next, the device models of the tachometer and the attitude control system confirm or deny this hypothesis. If the tachometer were stuck at 0, the attitude control system would react by adjusting the speeds of the other wheels to compensate for the perceived loss of momentum. However, since the wheel speed is not actually 0, the attitude control system would not have a correct estimate of the momentum. Changing the speed of the other wheels would result in disturbing the attitude of the satellite. Therefore, the hypothesis that the tachometer is faulty is denied and the heuristic which

suggested the hypothesis is modified to prevent suggesting the hypothesis in future similar cases. The conditions which were necessary to explain why the hypothesis failed are exactly those which should be added to the heuristic to avoid a hypothesis failure in similar cases. In this example, the heuristic is modified to test that the attitude is disturbed before postulating a faulty tachometer. Figure 10 contains the modified heuristic.

#### Figure 10: Revised Faulty Tachometer Heuristic- changes <u>underlined</u>.

The primary difference between explanation-based learning and inductive learning is how the class of "similar" failures is discovered. The goal is to define those failures which have a wheel speed of 0 but are not a broken tachometer. In explanation-based learning, this class is found to be those failures which do not have an attitude disturbance after the wheel speed reaches 0. This conclusion is deduced from a model of the connectivity and functionality of the components of the satellite. In contrast, in inductive learning, this class would be arrived at by correlating features of a number of positive and negative training examples. Since the satellite has over 300 signals in the attitude control system alone which are sampled every second, the time and space demands of correlating these features may make inductive learning impractical.

(After the tachometer rule is revised, ACES correctly identifies the fault to be in the wheel drive motor. The second rule in Figure 9 suggests that the pitch-roll+ wheel drive motor is not responding to its drive signal. The device model confirms this fault. This fault accounts for the pitch-roll+ wheel speed going from -200 to 0 as the results of friction. The speed of the other three wheels are automatically adjusted by the attitude control system to conserve the momentum. This occurs without any disturbance of the attitude.)

## Results

There are two standards for evaluating the effects of learning in ACES. First, there is the performance of ACES using the rules in Figure 9. We call this version naive-ACES. Additionally, there is the performance of ACES using rules hand-coded from information provided by an expert. We call this version of the system expert-ACES.

The data in Figure 11 demonstrate that the learning technique presented in this paper improves the simple fault diagnosis heuristics to the extent that the performance of ACES using the learned heuristics is comparable to the system using the rules provided by an expert. In one case, the performance of the learned rules is even better than the expert provided rules.

CASE	fault	naive-ACES	naive-ACES after learning	expert-ACES
1	tachometer	21	1	1
2	wheel drive-1	4	1	2
3	wheel unload	1	1	1
4	wheel drive-2	2	1	1

Figure 11: Number of Fault Hypotheses.

# **Economic Sanctions**

Our objective in constructing OCCAM is to build an knowledge-based system which predicts the outcome of applying economic sanctions. However, instead of heuristics rules, OCCAM relies on a memory of generalizations built when analyzing the outcome of previous incidents. OCCAM starts with general knowledge about coercion, and domain knowledge about political and economic relationships. OCCAM builds specializations of coercion which represent such events as blackmail, kidnapping<sup>5</sup> and economics sanctions. These specializations serve as schemata<sup>6</sup> which allow OCCAM to easily make a prediction when a situation is recognized as similar to previous situations (Schank, 1982). For example, consider answering the following question:

What might happen if the United States refuses to sell computer equipment to South Korea unless South Korea improves its record on human rights?

An expert on economic sanctions or someone familiar with this sort of incident might reply:

South Korea will probably buy computers from some other country such as Japan in the same manner that the USSR imported grain from Argentina when the US refused to sell grain to the USSR.

On the other hand, someone with no knowledge of previous incidents might arrive at a similar conclusion through a long inference chain: South Korea will have an increased demand for computers; South Korea would be willing to pay a higher price for computers; There are several countries which export computers; South Korea may buy the computers from one of them. In OCCAM, this sort of reasoning is done when a new event is added to memory, saved as a new schema and relied on to answer questions. For example, OCCAM should generate an answer to the South Korea question by accessing a generalization which represents those economic sanction incidents which have failed because the threat was to refuse to sell a product which was easily obtainable elsewhere. A primary question then is how should a schema be generalized from examples of economic sanctions. How can one determine which features of an economic sanctions incident are useful predictors of the outcome?

One way to determine which features are predictive is to use a similarity-based learning mechanism (Lebowitz, 1980) to determine which features are always present for each outcome. The problem with this approach is that when there are a large number of features, it requires a large number of examples to distinguish those similarities which are coincidental from those which are relevant. For example, consider the following two kidnapping examples:

#### Kidnapping-1

John was abducted. His father, a wealthy, fair skinned man, received a note which stated John would be killed unless he paid a \$100,000 ransom.

#### Kidnapping-2

Mary was abducted. Her mother, a wealthy, fair skinned woman, received a note which stated Mary would be killed unless she paid a \$50,000 ransom.

There are a number of similarities between these two kidnappings. In both instances, the parent who received the ransom note is wealthy and has fair skin. It is relevant that the ransom note be sent to a parent rather than just any person, since parents have a goal of preserving the health of their children. It is relevant that the parent is wealthy, since wealthy persons have the ability to pay the ransom. However, it is just a coincidence that the parents in these two kidnappings have fair skin.

The distinction between relevant and coincidentally similar features is important in making inferences and predictions. For example, if the only examples of kidnapping encountered are Kidnapping-1 and Kidnapping-2 and an intelligent person hears about another kidnapping he might want to infer that the

<sup>&</sup>lt;sup>5</sup>In the kidnapping domain, the domain knowledge includes interpersonal relationships (Pazzani, 1985).

<sup>&</sup>lt;sup>6</sup>OCCAM's schemata can be thought of as a special type of chunk (Laird et al., 1984, Larkin et al., 1980) which record causal and motivational information for events.

parent of the hostage is wealthy. On the other hand, he would not want to infer that the parent has fair skin. Similarly, if presented with a new kidnapping case where the parent has fair skin but is not wealthy, an intelligent person would not want to predict that the ransom would be paid. To avoid making erroneous inferences, a schema should not include all features which are common to all experienced events.

One way to avoid the problem of coincidentally similar features is to have an *a priori* set of features which are relevant (Kolodner, 1984). However, in general, the relevance of features is dependent on the type of event. Consider the following examples:

#### Cancer-1

John, a wealthy, fair skinned man, was advised by his doctor to wear a 15 SPF sunscreen at the beach.

#### Cancer-2

# Mary, a wealthy, fair skinned woman, was advised by her doctor to wear a 15 SPF sunscreen at the beach.

In these examples, fair skinned would be considered relevant since fair skinned persons are more prone to skin cancer. The fact that Mary and John are both wealthy is incidental. These examples illustrate that the relevance of feature is not absolute. A key task for a learner, either human or computer, is identifying which features of a particular event should be expected to appear in future events.

When a proper domain theory exists, explanation-based learning offers a solution to the problem of identifying relevant features. It can determine the relevant features from analyzing only one example. The relevant features are precisely those features which enable the causal and intentional relationships to be inferred. The explanation-based learning approach used in OCCAM is very similar to that used by ACES. An explanation is constructed which explains why a certain event occurs. For OCCAM, the explanation indicates why an economic sanctions incident succeed or failed. In ACES, the explanation indicates why a hypothesized fault has been ruled out. The explanation is generalized by retaining only those parts of the example which were needed to produce the explanation. The generalized example and the explanation are saved so that a single step (or chunk) can be substituted for a longer inference chain.

For some problems, it is important that a schema be created to avoid extremely complex searches. One situation in which this occurs is when the motivation for a particular course of action is to avoid a potential goal failure. It is difficult, if not impossible to determine why the action was decided upon without knowing about the potential failure. For example, OCCAM notices a similarity between a number of kidnappings whose victims are all infants with blond hair. However, it is not capable of finding an explanation for this similarity until it finds a kidnapping case in which there is a goal failure when the victim was not an infant. After the ransom was paid, the victim was able to give people evidence which led to the arrest and conviction of the kidnapper (which is, of course, a goal failure for the kidnapper.) With this new information, an explanation for the kidnapping of infants is found to be avoiding this goal failure since infants cannot give police evidence nor testify.

In the next section we give an example of OCCAM creating a schema with explanation-based learning. In addition to the explanation-based learning module, OCCAM consists of a memory indexing scheme which enables appropriate examples and generalization to be found in memory, and a similarity-based learning mechanism which can learn some of the domain theories used by the explanation-based learning system. The interested reader is referred to (Pazzani, 1986b) and (Pazzani, 1985). Here, we concentrate on only the explanation-based learning module.
## Economic Sanctions: An Example

Consider the following example of economic sanctions:

### **Economic-Sanctions-1**

In 1983, Australia refused to sell uranium to France unless France ceased nuclear testing in the South Pacific. France paid a higher price (\$300,000,000) to buy 1500 tons of uranium from South Africa.

What lesson should be learned from Economic-Sanctions-1? That economic sanctions never achieve their desired goal? This conclusion would be overly general. That economic sanctions never work against countries that exports wine? This conclusion would be wrong. That economic sanctions won't work when Australia refuses to sell a country a commodity which is sold by South Africa. This is close to being true, but is probably too specific to apply to many future cases. The problem here is to identify which features of Australia, South Africa, France, and uranium as well as the features of demand (to stop nuclear testing in the South Pacific) and the threat (to not sell uranium) were necessary to explain why Australia did not achieve its goal. To get an idea of the magnitude of the problem consider the representation of Economic-Sanctions-1 which is used in OCCAM (see Figure 12).

In Figure 12, the Conceptual Dependency representation of Economic-Sanctions-1 is displayed. The top level concept is coerce. In addition to economic sanctions, coerce underlies the representation of blackmail and kidnapping. coerce has a number of roles: an actor who performs the coercion; a target which is the victim of the coercion; an object which is the focus of the coercion (in kidnapping, the object is typically a relative of the target, in economic sanctions, the object is usually a commodity); a demand which is an action the actor wants the target to perform; a threat which the actor will perform if the target doesn't meet the demand; a response which the target performs in response to the threat; and a result which is the outcome of the coercion. In Figure 12, the notation (\*role\* target) indicates that this component is identical to the target of the coerce.

The information in OCCAM's representation about individual countries is derived from the World Almanac. For example, for South Africa, which is the actor of the response, the language is English, the location is the southern hemisphere, it has a business relationship with the US, Japan, Australia (the actor), and France (the target), the continent is Africa, it exports uranium (the object), gold, chromium and diamonds and it imports oil. In general, the approach in OCCAM has been to provide as many features of each action and entity as possible. In fact, some features which OCCAM uses, such as the business relationships of France and Australia were not included in Figure 12 because they would not fit in the figure. It is an important part of the generalization process to determine which of the features should be included in a generalization

When generalizing Economic-Sanctions-1, some features of entities such as South Africa are relevant, other's aren't. It is the simple economic theories used by OCCAM which indicate the relevancy of features. Only those features which are needed to establish an explanation of why France was able to buy uranium from South Africa are considered relevant. A closer examination of the generalization procedure in OCCAM should help clarify how relevancy is determined.

The first step in generalizing is deciding if an event should be generalized at all. There are a number of criteria to determine if a single event is worth generalizing. DeJong (DeJong, 1983) gives a number of criteria to determine if a single event is worth generalizing (e.g., does the event achieve a goal in a novel manner). In this case, let's assume that there are no previous examples of economic sanctions in memory so when Economic-Sanctions-1 is encountered, a goal has failed in a novel manner. This goal failure initiates the explanation and generalization process. The goal failure for Australia was accomplished when France purchased uranium from South Africa. The explanation process tries to determine why France was able purchase the uranium. The rules in Figure 13 are used by OCCAM to create an explanation how France was able to purchase the uranium causing a goal failure for Australia.

```
(coerce object (commodity type (uranium)
                          amount (weight number (1500)
                                         unit (tons)))
       actor (country name (australia)
                       language (english)
                       location (southern-hemisphere)
                       government (parliamentary)
                       continent (australia)
                       exports (*set* (commodity type (wool))
                                       (*role* object)
                                       (commodity type (diamond)))
                       imports (*set* oil))
       target (country name (france)
                        language (french)
                        government (republic)
                        continent (europe)
                        location (northern-hemisphere)
                        imports (*set* (*role* object)
                                        (commodity type (oil)))
                        exports (*set* (commodity type (wine))
                                        (commodity type (cheese))))
       demand (act type (explode)
                    actor (*role* target)
                    object (weapons type (nuclear))
                    location (southern-hemisphere)
                    mode (neg))
       threat (act type (sell)
                    actor (*role* actor)
                    object (*role* object)
                    to (*role* target)
                    mode (neg))
       response (act type (sell)
                      actor (country name (south-africa)
                                     language (english)
                                     location (southern-hemisphere)
                                     business-relationship
                                      (*set* (country name (us))
                                             (country name (japan))
                                             (*role* actor)
                                             (*role* target)
                                             (country name (uk)))
                                     government (parliamentary)
                                     continent (africa)
                                     exports
                                      (*set* (*role* object)
                                             (commodity type (gold))
                                             (commodity type (chromium))
                                             (commodity type (diamond)))
                                     imports (*set* (oil)))
                      object (*role* object)
                      price (money dollars (30000000)
                                   value (>market))
                      to (*role* target))
       result (state type (possess)
                      actor (*role* target)
                      value (yes)
                      object (*role* object)))
```



```
(def-rule refuse-to-sell->demand-increase
        (act type (sell)
               actor (country exports ?y)
               TO ?x
               object ?y
               mode (neg))
 result
  (state type (demand-increase)
         actor ?x
         value (yes)
         object ?y))
(def-rule demand-increase->price-increase
          (state type (demand-increase)
                 value (yes)
                 actor ?x
                 object ?y)
 enables
  (act type (sell)
      actor (country exports ?y
                      business-relationship ?x)
      to ?x
      object ?y
      price (money value (>market))
      mode (yes)))
(def-rule sell->possess
          (act type (sell)
               to ?x
               object ?y
               mode (yes))
 result
 (state type (possess)
        object ?v
        value (yes)
        actor ?x))
```

### Figure 13: OCCAM's economic rules.

The first rule in Figure 13 indicates that the result of Australia refusing to sell uranium to France is that France will have an increased demand for uranium. The second rule allows OCCAM to infer that the increased demand for uranium will enable a country that exports uranium and has a business relationship with France (i.e., South Africa) to sell France the uranium at a price greater than the market price. Finally, the third rule indicates that selling the uranium to France will result in France having the uranium. The components of the representation of Economic-Sanctions-1 which correspond to the components of the rules will be included in the generalization of Economic-Sanctions-1. Everything else is irrelevant and will be discarded. The generalization produced by OCCAM is illustrated in Figure 14.

The generalization in Figure 14 indicates that if a country that exports a commodity tries to coerce a country which imports the object by refusing to sell them the commodity, then a response might be to buy the commodity at a higher price from another country. Although this seems like a simple conclusion, there are many examples where economic sanctions have failed for this reason (Hufbauer and Schott, 1985) (e.g., in 1961, the USSR refused to sell grain to Albania, who purchased it instead from China, in 1980, the US refused to sell grain to the USSR who purchased it instead from Argentina, and in 1981, the US refused to sell pipeline equipment to the USSR who purchased it instead from France).

A similar conclusion about the effectiveness of sanctions was arrived at by Ian Smith, the former Prime Minister of Rhodesia which was the target of a decade of economic sanctions following it's indepedence

(coerce result (state type (possess) actor (\*role\* target) object (\*role\* object) value (yes)) response (act type (sell) actor (country exports (\*role\* object) business-relationship (\*role\* target)) object (\*role\* object) to (\*role\* target) price (money value (>market))) threat (act type (sell) actor (\*role\* actor) object (\*role\* object) mode (neg)) target (country imports (\*role\* object)) object (commodity) actor (country exports (\*role\* object)))

Figure 14: Generalization of Economic-Sanctions-1 produced by OCCAM.

from Great Britian (cited in (Renwick, 1981)):

We find that we are compelled to export at discount and import at a premium.

It is interesting to note which features of South Africa were generalized. In Figure 14, the only features of the actor of the response are that it exports the object and it has a business relationship with the target. Notice that in the individual episode in Figure 12, the fact that South Africa has a business relationship with the actor was also included. However, this feature was not necessary to produce the explanation, so it is not included in the generalization. The features of South Africa which were included are those that matched against the features of the second rule in Figure 13.

The generalizations produced by OCCAM serve as a means of recognizing when economic sanctions will fail or succeed at achieving the desired goal. These generalizations summarize a inference chain by retaining only those surface features of an example which were required to establish an explanation for the outcome using simple economic theories.

## Conclusion

We have demonstrated that explanation-based learning is a viable technique for obtaining expert-like performance in an knowledge-based system. Explanation-based learning can be used in knowledge-based systems where the heuristics to be learned are implicit in a different representation of the same knowledge. For example, in fault diagnosis, device models are a natural way of expressing the functionality of a component. However, they are not the most natural or efficient representation for diagnosis (Sembugamoorthy and Chandraskaran, 1985). The ACES system explicates associations between failures and atypical behavior which are implicit in device models. In OCCAM, explanation-based learning creates schema which recognize when economic sanctions will fail or succeed at achieving the desired goal. This schemata explicate the implications of simple economic theories such as supply and demand on economic sanctions incidents.

# Acknowledgements

The research on ACES was supported by the Aerospace Sponsored Research Program. The research on OCCAM was supported in part by the UCLA-RAND Artificial Intelligence Fellowship. Carl Kesselman implemented the PROLOG meta-interpreter and commented on an earlier draft of this paper.

## References

- Berl, J., Lewis, G. and Morrison, R. Applying Models of Choice to the Problem of College Selection. In Carroll, J. and Payne, J. (Ed.), *Cognition and Social Behavior*. Hillsdale, NJ: Erlbaum, 1976.
- Charniak, E., Riesbeck, C. and McDermott, D. Artificial Intelligence Programming. Hillsdale, NJ: Lawrence Erlbaum Associates, 1980.
- Davis, R., Shrobe, H., et al. Diagnosis Based on Description of Structure and Function. In *Proceedings* of the National Conference on Artificial Intelligence. Pittsburgh, PA: American Association for Artificial Intelligence, 1982.
- de Kleer, J. and Brown, J. A Qualitative Physics Based on Confluences. *Artificial Intelligence*, 1984, Vol. 24(1).
- DeJong, G. Acquiring Schemata Through Understanding and Generalizing Plans. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*. Karlsruhe, West Germany: , 1983.
- Ericsson, K and Simon, H. Protocol Analysis: Verbal Reports as Data. MIT Press, 1984.
- Forbus, K. Qualitative Process Theory. Artificial Intelligence, 1984, Vol. 24(1).
- Genesereth, M., Bennett, J.S., Hollander, C.R. DART: Expert Systems for Automated Computer Fault Diagnosis. In *Proceedings of the Annual Conference*. Baltimore, MD.: Association for Computing Machinery, 1981.
- Hufbauer, G.C., and Schott, J.J. *Economic Sanctions Reconsiderd: History and Current Policy*. Washington, D.C.: Institute For International Economics, 1985.
- Kolodner, J. Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model. Hillsdale, NJ.: Lawrence Erlbaum Associates, 1984.
- Kuipers, B. Commonsense Reasoning about Causality: Deriving Behavior from Structure. Artificial Intelligence, 1984, Vol. 24(1).
- Laird, J., Rosenbloom, P., and Newell, A. Towards Chunking as a General Learning Mechanism. In *Proceedings of the National Conference on Artificial Intelligence*. Austin, Texas: American Association for Artificial Intelligence, 1984.
- Larkin, J., McDermott, J., Simon, D., and Simon, H. Expert and Novice Performance in Solving Physics Problems. *Science*, 1980, *208*, 1335-1342.
- Lebowitz, M. *Generalization and Memory in an Integrated Understanding System* (Computer Science Research Report 186). Yale University, 1980.
- Michie, D. Inductive Rule Generation in the Context of the Fifth Generation. In *Proceedings of the* International Machine Learning Workshop. Monticello, Illinois: , 1983.
- Minton, S. Constraint-based Generalization: Learning Game-Playing Plans from Single Examples. In Proceedings of the National Conference on Artificial Intelligence. Austin, TX: AAAI, 1984.
- Mitchell, T., Kedar-Cabelli, S. and Keller, R. A Unifying Framework for Explanation-based Learning. *Machine Learning*, 1986, Vol. 1(1).
- Mitchell, T., Mahadevan, S., and Steinberg, L. LEAP: A Learning Apprentice for VLSI Design. In *International Meeting on Advances in Learning*. Les Arc, France: , 1986.
- Nelson, W.R. REACTOR: An Expert System for Diagnosis and Treatment of Nuclear Reactor Accidents. In *Proceedings of the National Conference on Artificial Intelligence*. Pittsburgh, PA: AAAI, 1982.
- Nisbett, Richard and Ross, Lee. Human Inference: Strategies and Shortcomings of Social Judgements. Engelwood Cliffs, NJ: Prentiss-Hall, Inc., 1978.

- Nisbett, R., and Wilson, T. Telling More Than We Can Know: Verbal Reports on Mental Processes. *Psychological Review*, 1977, Vol. 84(3).
- Pazzani, Michael. Explanation and Generalization-based Memory. In *Proceedings of the Seventh* Annual Conference of the Cognitive Science Society. Irvine, CA: Cognitive Science Society, 1985.
- Pazzani, M. Refining the Knowledge Base of a Diagnostic Expert System: An Application of Failure-Driven Learning. In *Proceedings of the National Conference on Artificial Intelligence*. American Association for Artificial Intelligence, 1986.
- Pazzani, M., Dyer, M. and Flowers, M. The Role of Prior Causal Theories In Generalization. In *Proceedings of the National Conference on Artificial Intelligence*. American Association for Artificial Intelligence, 1986.
- Pazzani, M. and Brindle, A. An Expert System for Satellite Control. In *Proceedings of ITC/USA/85, the International Telemetering Conference.* Las Vegas, NV: International Foundation for Telemetering, 1985.
- Pazzani, M. and Brindle, A. Automated Diagnosis of Attitude Control Anomalies. In *Proceedings of the* Annual AAS Guidance and Control Conference. Keystone, CO: American Astronautical Society, 1986.
- Renwick, Robin. Economic Sanctions. Center for International Affairs, Harvard University, 1981.
- Scarl, E.A., Jamieson, J., and Delaune, C. A Fault Detection and Isolation Method Applied to Liquid Oxygen Loading for the Space Shuttle. In *Proceedings of the Ninth International Joint Conference* on Artificial Intelligence. Los Angeles, CA: , 1985.
- Schank, R. Dynamic Memory: A Theory of Reminding and Learning in Computers and People. Cambridge University Press, 1982.
- Sembugamoorthy, V. and Chandraskaran, B. Functional Representation of Devices and Compilation of Diagnostic Problem Solving Systems (Tech. Rep.). Ohio State University, March 1985.

Shortliffe, E.H. Computer-based Medical Consultation: MYCIN. New York, NY: American Elsevier, 1976.

- Steels, Luc and Van de Velde, Walter. Learning in Second Generation Expert Systems. In J.S. Kowalik (Ed.), *Knowledge-Based Problem Solving*. Engelwood Cliffs, NJ: Prentiss-Hall, Inc., 1986.
- Wagner, R.E. Expert System for Spacecraft Command and Control. In *Computers in Aerospace IV Conference.* Hartford, CT: American Institute of Aeronautics and Astronautics, 1983.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

## ANALYSIS OF THE PERFORMANCE OF A GENETIC ALGORITHM-BASED SYSTEM FOR MESSAGE CLASSIFICATION IN NOISY ENVIRONMENTS

Elaine J. Pettit	Michael J. Pettit
Merit Technology, Inc.	Dept. of Computer Science
17770 Preston Road	Univ. of Texas at Dallas
Dallas, TX 75252	Richardson, TX 75083

#### ABSTRACT

The process of knowledge acquisition must occur continually in those knowledge-based systems which must operate in noisy, contextually rich environments. One very important application with this requirement involves the inferring of the occurrence of events which cannot be exhaustively predefined from variably noisy sensor messages. Our paper describes on-going basic research for construction of an adaptive system which can perform high-level, rapid classification of sensor messages, possibly very noisy, concerning objects in its environment. The paper concentrates on experiments to determine optimal parameters for this bi-level, genetic algorithm-based system in low, medium, and high noise environments.

### I INTRODUCTION AND MOTIVATION

One of the most stubborn areas of knowledge acquisition for knowledge-based systems has been the construction of a system which can acquire knowledge during its field operation. This is particularly true of many military applications in environments which preclude the umbilical cord to the knowledge engineer that most knowledge-based systems require. Many of these applications involve the encoding and classification of messages (from sensors, intelligence, or direct observation) which a human analyst must integrate and then attempt from this fused information to infer the tactics of his adversary. Variations of this process are known as situation assessment, threat assessment, information fusion, and tactics analysis.

There are at least two roles which machine learning may play in such knowledge-based systems: 1) knowledge acquisition in the building stages of the knowledge-based system (ab initio knowledge acquisition), and 2) knowledge acquisition in the operational system (in situ knowledge acquisition). The first has received considerable attention since the popularization of expert systems has underscored the problems, both in time and quality, of getting information from experts into the system. In many ways, however, the solution of this problem is of con-

siderably less benefit over the long term than successfully implementing in situ knowledge acquisition. Although the verification of the completeness and consistency of the initial knowledge base is vital to knowledge-based systems, there is no escaping the fact that everything a system will eventually encounter cannot be foreseen at the time of its creation. Without the capacity to deal with unknown situations, and even more importantly, to sieve meaningful information in the presence of noise, a real-world knowledge-based system will show sharp degradation under these conditions. This phenomenon of system brittleness is well-recognized in expert system technology and has been described as "falling off the knowledge cliff" (Feigenbaum as cited in Michalski, 1986). Although the use of pre-defined generic scenarios can be useful here, without the ability to extend and refine those scenarios beyond simple instantiation the system will not be able to offer assistance when the need is most critical: when unanticipated obstacles, inoperative equipment, contradictory information, and loss of active personnel combine to present scenarios that are outside the scope of those originally envisioned. Even for routine military or civilian missions, the more closely the system knowledge resembles the real world, the more timely and reliable its advice will be.

There exist some heuristics learning systems which could possibly, with modification, exhibit on-site adaptability. However, for situation assessment problems, satisfaction of the requirement for adaptability alone is insufficient: noise immunity is critical. The process of inferring the occurrence of events from messages produced by sensors and other sources is subject to noise from two sources: 1) noise in the attributes of the patterns reported by a given sensor (a characterization problem); and 2) noise, due to the asynchronous nature of message traffic, in the number of patterns reported by sensors in a given time step (a fusion problem). Thus, an automated system for classifying these messages into meaningful events must remain immune to noise at two levels of a classification hierarchy: the message level and the event level. It must first be able to characterize accurately an event as it is occurring; secondly, it must be able to distinguish between noise-induced variations of an ongoing event versus transition to a new event The problem is one of both knowledge representation and state. knowledge utilization, as described in Section II, "Related Work".

To address the above issues, we are currently researching a hybrid methodology for knowledge representation and operation built upon genetic algorithms (Holland, 1975) and conceptual clustering (Michalski and Stepp, 1983). Conceptual clustering provides a means for representing and operating upon knowledge

so that synergistic properties are expressible and a spectrum of deviations are tolerated. Genetic algorithms provide a strategy for highly effective pseudo-parallel search of feature hyperspace with noncommensurate, ill-behaved optimization criteria. As an initial step, we have implemented a genetic algorithmbased message classification system to explore how well it resolves the following issues:

> 1) How sensitive is it to managing noise in a stable environment versus recognizing when transition to a new environment has occurred? At what point do we alter not only the low-level message input classifiers but also the higher level "conceptual classifiers" that aggregate these messages into meaningful categories? (To what degree and in what functions should incremental learning occur and when should it be postponed?)

> 2) Assuming a bi-level system architecture driven by genetic algorithms and operating with limited resources, should the rate of change in event recognition classifiers lag considerably behind that of message classifiers or vice versa? Increasing the rate of adaptation of message classifiers would allow recognition over noise at the cost of nonconvergent event classification at first: increasing the rate of adaptation of event classifiers would allow consideration of all possible interpretations quickly but with a loss of accuracy. At what point is there a tradeoff?

Concommitant with the resolution of these application issues was the need to extend, with sound theoretical underpinnings, the basis of the genetic algorithms themselves to meet certain implementation requirements. Primary among these was the need to define composite, variable length classifiers (to represent the event classifiers) and genetic operators and performance measures appropriate to these kinds of classifiers. There was a secondary need to devise a means of handling several different ongoing patterns (each of the noisy messages) in the message classifier population.

Because the research base of using genetic algorithms in stochastic environments is very sparse, we have concentrated in this paper on the definition and development of a scientific framework for empirically testing with appropriate statistical measures the behavior of various models postulated to meet the above requirements. We have begun our investigation with a "worst case" scenario: noise induction is random, adaptation cycles are below empirically derived "good" limits, and we use "offline performance" (described in Section III) which is the least discriminating (between control and experimental performance) but probably the most realistic measure of algorithm performance in this task domain. We feel those strengths that manifest themselves under these conditions will provide the most fruitful areas of future research. We present our framework, including details of our data generation and analysis, and our research findings to date concerning the efficacy of genetic algorithms in noisy message classification.

## II RELATED WORK

There are currently two primary approaches to dealing with the problems of noise and adaptability in the development of knowledge-based systems. One is to define a set of generic objects and the processes governing their behavior, instantiate those objects with values derived from the situation, and operate on those objects in accordance with some predetermined inference strategy for resolving conflicts and focusing ambiguity. This approach is basically differential in nature, treating the "instantaneous" differences between data items or between data items and a predetermined template as the modeling basis. Several artificial intelligence techniques have been proposed to address the problem of handling preconditions established by incomplete, ambiguous, or, in the presence of an intelligent adversary, deceptive data. Most readily applicable are those which deal formally with nonmonotonic reasoning and uncertainty. These include Bayesian inference, fuzzy logic, non-monotonic logic, belief revision, endorsement theory, Dempster-Shafer theory of evidence, and cautious inference. The end result of each of these methods is that snapshots of parts of the environment are pieced together by predefined evidential reasoning algorithms to create the most probable overview at that time.

These methods quantify the contribution of the value of a data item AFTER a match has been established. What these methods cannot capture is the epistatic interaction AMONG data items in 1) establishing relevance of data at a conceptual level to the problem (as opposed to matching individual data items); and 2) reinforcing or negating the individual contributions of data items taken in all possible combinations. The problem is formally isomorphic to the optimal allocation of trials in a search of feature hyperspace for which the probability distributions of "success" and "failure" are unknown (and because of distribution overlap, empirically indecidable) for individual dimensions ((Holland, 1975) and (DeJong, 1975) provide an excellent discussion of this "k-armed" bandit problem).

Several of the foremost areas in machine learning address these problems of incorporating adaptability and noise immunity into knowledge representation and utilization paradigms. Michalski's INDUCE and CLUSTER paradigms (Michalski, 1983) (Michalski and

Stepp,1983) have demonstrated noise tolerance in training examples. However, these algorithms employ a search tapering technique based on best-first search which, from our analysis, may prove inefficient, and at worst, non-robust, in the complexity of the situation assessment domain.

Quinlan's ID3 (Quinlan,1986) and its progeny ID4 (Schlimmer and Fisher,1986) use a statistic based on object occurrence probability with a chi-square distribution to determine which attributes should be rejected in constructing classification decision trees. However, both these systems are nonincremental, i.e., they require all the objects to be classified to be available at one time for iterative processing. They are therefore not applicable to tracking asynchronous message traffic over time. Furthermore, although Quinlan's empirical evidence indicates some utility for this statistic, one must question its use for dependent attributes since the statistic assumes independence among the variates.

Schlimmer and Granger (1986) have recently presented research which, of the alternate systems examined, is most directly applicable to the message classification problem. Their STAGGER system is designed to track "concept drift", which, as in our system, is the ability to distinguish between local noise in attributes and a global change of events. It uses Bayesian propagation of likelihood ratios to incrementally refine weights associated with individual characterizations and the structure of the characterizations themselves. The primary theoretical difference between the STAGGER method and the use of genetic operators lies in the reliance of STAGGER on the robustness of the incrementally derived probabilities to characterize what we state is an indecidable probability distribution.

Most prior research on the use of genetic algorithms themselves has concentrated on their use in a non-stochastic environment, i.e., where noise in the patterns to be tracked and learned is minimal and the patterns themselves occur consistently throughout the tracking period. Booker (1985) has studied the optimization of parameter settings for genetic algorithm-based message classifier systems in non-noisy environments: his research has served as a strong basis for the parameter settings in our system, as well as forming a basis for our application of multiple pattern tracking in the message classifier population. Smith's genetic algorithm-based poker playing system (Smith, 1980) was highly successful against Waterman's system (Waterman, 1975): he and (Grefenstette et al., 1985) have supplied a basis for hierarchical composition of genetic operators. There have also been numerous successes in applying these theories in areas as diverse as dynamic system control (Goldberg, 1983), function optimization (Bethke, 1980), and object movement in a complex environment (Holland, 1986). Previous research by (Pettit and Swigger, 1983)

has indicated that even in noisy environments, a genetic algorithm-based system can track the pattern of an environment in realistic, correlated flux more successfully than a computationally prohibitive model which maintains detailed statistical information regarding the status of each component.

#### III EXPERIMENTAL METHODOLOGY

#### III.1 Background on Genetic Algorithms

In one of the first mathematical analyses of the use of biologically based mechanisms in promoting adaptation in artificial systems, John Holland (Holland, 1975) reviewed the prolonged success of lifeforms in adapting to an environment through evolution. The biological organism is faced with testing a large set of possible genetic expressions in its offspring by means of environmental interaction with a relatively small subset of realized structures (its own genotype). Nonlinearity and epistatic interactions among gene sets complicate the problem of achieving a successful, if not optimal, genetic complement in offspring.

Holland has mathematically hypothesized that genetic operators (e.g., crossing-over, mutation, inversion) exploit the optimization of reproductive fitness (number of offspring) by a means he terms intrinsic parallelism. Intrinsic parallelism is the testing of a large pool of schemata - i.e., the set of all partitions and combinations thereof of a prototypical structure - by means of a much smaller subset of realized structures. More simply, consider the structure A consisting of a string of six binary digits, (1 0 1 1 0 1). Each binary digit may be considered to be a classifier of a binary feature vector (such as the premises in a rule-based system). Structure A is a member of a set of structures ALPHA which includes all possible strings of six binary digits. There exists a superset EPSILON which is the set of strings of length six composed of concatenations of {1,0,#}, where # represents a "don't care" position, i.e., its value as 0 or 1 is irrelevant. For example, let E be an element in EPSILON of the form (1 0 # # 0 This "E" is termed a schema, and all possible schemata com-1). pose EPSILON, the "pool of schemata". The occurrence of structure A, then, is a sample not only of A itself but of the schemata  $(1 \ 0 \ \# \ \# \ 0 \ 1)$ ,  $(\# \ \# \ \# \ 0 \ 1)$ ,  $(1 \ 0 \ 1 \ 1 \ 0 \ \#)$ , etc. Thus, parallel sampling of applicable schemata can be implemented with relatively few templates.

Now consider a structure ENVIRONMENT (0 0 1 0 0 1) which represents the "state" of an environment. One - and by no means the only - measure of the fitness of Structure A in characterizing ENVIRONMENT is a computation of a metric of difference (e.g., Hamming distance) between A and ENVIRONMENT. Schemata thus represent the contribution to fitness of single detectors as well as of combinations of detectors. A subset of structures from ALPHA constitutes a population. It is, by definition, the goal of adaptation to modify these structures in order to optimize the fitness of the population. Holland and others have shown that genetic operators such as crossing-over are highly successful in 1) testing a large number of possible schemata through modifications on a much smaller number of realized structures, and 2) exploiting local optima on the way to achieving the global optimum without becoming entrapped (as often happens in hill-climbing optimization techniques).

A general algorithm for genetic adaptation is given below. Algorithms for the genetic operators themselves are presented under Section III.3, "Experimental Design".

### GENERAL ALGORITHM FOR GENETIC ADAPTATION

Initialize population (e.g., message classifiers) and environment. Find the performance for each structure in the population and call it MU(i). Define the random variable RAND on  $\{1,\ldots,M\}$  by assigning probability  $(MU(i)/\sum_{j=1}^{M} MU(j))$  to the ith structure in the population, where M is the number of structures in the population. Make M trials of RAND, each time storing the structure at position RAND in the population at successive positions in auxiliary list TEMP-POP. For each structure in TEMP-POP, apply mutation and other unary genetic operators with some predefined probability, and, if it is a structure in an even-numbered position, perform crossover between it and its immediate predecessor. Set the original population to TEMP-POP, and repeat all steps except initialization for the desired number of generations.

As stated in the preceding section, it has been demonstrated that such a model, with the addition of suitable selection criteria and control structures, can track the pattern of an environment in flux more successfully than a computationally prohibitive model which maintains detailed statistical information regarding the status of each component (Pettit and Swigger, 1983). At the same time the genetic model does not discard instances of new schemata when an optimum is obtained, allowing for recovery over another set of absorbing transitions when the pattern is altered in a realistic, correlated fashion. In contrast, it is possible for a component-sampling model to lock into a present optimum that was maintained over sufficient transitions: the capability for change would eventually become miniscule.

### III.2 System Overview

Figure III.2-1 illustrates our general system design. The system resides in an environment in which objects of four types come and go. Events are the power set of the set of four object types exclusive of the null set and inclusive of the improper subset (i.e., no more than one object of one type can be present, although an extension of the system has been designed to handle this case in future research). The objects post messages concerning their type and attributes to the environmenal interface board. Noise via Markov chain transition matrices are induced in the values of these messages at an adjustable level. The noisy messages, ranging from 1 to 16 per object, are posted to an intermediate sensory board.

The message classifier system then uses genetic algorithms to track the noisy messages as they are posted. The results of this tracking are posted to the message-to-event-classifiers interface board. The event classifier system then uses genetic algorithms to track event components from this board, i.e., number of objects in the environment and their types. The event classifier system posts its results to the output board. These results are compared to the actual environment for the performance measurement.



#### SYSTEM OVERVIEW

FIGURE III.2-1 : System Overview

### III.3 Experimental Design

Implementation Specification: All software is coded in ZetaLisp on Merit's corporate Symbolics 3670. The unoptimized compiled program consisting of five test cases and one control running under each of three noise levels with a global time period of 10 and a generation limit of between 200 and 400 iterations takes approximately 30 hours to run (since the data sets are of stochastically variable length, runtime varies). It should be noted that Holland's original design was intended for a parallel implementation. Since the paradigm involves primarily independent iterations, runtime for a parallel implementation should decrease in at least a roughly linear proportion to the number of processors available.

Functional Description: The following algorithm describes the general functional operation of the system. The algorithm was adjusted as necessary to fulfill the conditions of each experiment and the control.

```
function main:
   generate an initial population (size 50) of message
      classifiers;
   generate an initial population (size 50) of event
      classifiers;
   for time-step from 1 to global time limit do
  begin
     generate new set of environmental objects [1..4 objects
        of type 1, 2, 3, 4];
     set variability modulus to desired level;
     for message genetic timestep from 1 to generation limit
        do
    begin
        induce message noise and post noisy messages to
           sensory board (1 to 16 per object);
                      /* process through message classifiers */
        calculate the performance (P1) of each message
           classifier against each message on the
           sensory board to form subpopulations (one
           subpopulation of 50 classifiers per entry on
            sensory board);
        apply genetic algorithms to subpopulations;
        select from each subpopulation on the basis of
           performance (Pl) against that subpopulation's
           basis message an equal proportion (1/no. of sensory messages) of individuals to become the 50
           message classifiers of the next generation;
        if (message genetic timestep mod variability modulus =
           0) or (variability modulus < 1) then
                 /* process through event classifiers */
        begin
```

select best individual from each message group
 and post to message-to-event-classifiers blackboard;

calculate the performance (P2) of each event classifier against each message on the message-to-event-classifiers blackboard; apply genetic algorithms to the event classifier

population;

end /\* process through event classifiers \*/ end /\* genetic time loop \*/

calculate performance (P3) of each event classifier to .actual environment to get performance measure at current time\_step;

end /\* global time loop \*/. end function main.

•

Data Item Description: Objects in the environment consist of length 21 binary strings over the alphabet {0,1} with reserved bits i mod 10 = 0 or 1 indicating object type.

A random number, between 1 and 16 inclusive, of "noisy messages" are produced for each object by inverting the value of each bit with a probability of \*noise-level\*. The \*noise-level\*s used for this set of experiments were 0.25, 0.125, and .0625, signifying high, medium, and low levels respectively. Noise induction by this method creates a "worst-case" model of random noise, as opposed to a correlated, realistic flux (the "systematic" noise of Schlimmer and Granger, 1986). These messages are posted to the \*sensory-board\*.

The first level of the system consists of a population of size 50 of message-classifying strings of length 21 over the alphabet {0,1}. A population of size 50 has been shown by (Booker,1985) and (DeJong,1975) to demonstrate acceptable convergence in a non-stochastic environment.

These message-classifying strings are compared to the noisy messages posted to the \*sensory-board\* and are recombined in accordance with the general genetic adaptation algorithm. Performance for reproduction is computed using the M3 method of (Booker, 1985): this method, termed Pl for this model, is explained under the subtopic "Performance Measures".

The second level of the system consists of a population of size 50 of event-classifying composite strings in which the string elements are themselves strings of length 21 over the alphabet {0,1}. The composite strings consist of from 1 to 4 of these elemental strings, with only strings denoting different object types allowed in the composite.

These event-classifying strings are compared to the results posted on the message/event interface board by the message classifying strings and are recombined in accordance with the general genetic adaptation algorithm. Because of the composite nature of these strings, new genetic operators have been devised, based on the open and closed variability genetic regions postulated by evolutionary ecology (Carson, 1975) (Chapman, et al., 1979). Initial analysis indicates that these operators exhibit properties comparable to the intrinsic parallelism of (Holland, 1975). They are described further under the subtopic "Genetic Operators". Performance for reproduction is computed using a modified M3 method from (Booker, 1985), with modifications which retain the composite nature of the strings (see subtopic "Performance Measures").

After a preset number of generations, the best match score (see description of "P3" under "Performance Measures") for the current event-classifier population is output, and a new set of objects is generated. Each new set of objects constitutes a time-step, and \*global-time-limit\* number of these time-steps are performed.

#### Genetic Operators:

Noncomposite Operators (used in the message classifiers): a) Mutation: Make an equiprobable trial of a random variable X on {1,...,L}, where L is the number of positions in the structure undergoing mutation. Make an equiprobable trial of a random variable CHANGE on the integers 0,1. Replace the value of the element at position X with CHANGE. The rate of application of mutation for these experiments was .05, slightly higher than the .02 (1/population\_size) suggested by (DeJong, 1975), due to the system's operation in a stochastic environment (a higher mutation rate has been shown to slow allele loss). b) Crossover: Make an equiprobable trial of the random variable X on {1,...,L}, where L is the number of positions in each of the two structures undergoing crossover. If X = 0 or L, then no crossing-over takes place. Otherwise, take positions 1 through X of the first structure and append positions X+1 through L of the second structure. Likewise, take positions 1 through X of the second structure and append positions X+1 through L of the first structure. The rate of application of crossover for these experiments was 1.00, our rationale being that tracking in a stochastic environment would be improved with frequent exchange of pattern

parts.

No other unary genetic operators were used in the noncomposite case.

<u>Composite Operators</u> (used in the event classifiers): a) <u>Event-mutation</u>: Randomly select an element from the classifier and replace it with a new, randomly-generated instance of that object type. This is a highly disruptive operation, guaranteeing no preservation of patterns that may have been converging over several generations, but one which is critical to the avoidance of local optima. Mutation rate was maintained at .05.

Ł

b) <u>Event-crossover</u>: Three types were defined to attempt to replicate the effects of noncomposite crossover on the composite population. Some precedence for these definitions may be found in (Smith, 1980).

External crossover: Randomly select an element of the first structure. Randomly generate a set of object-types of which the object type of this element is a member. Exchange all elements of these object-types for which there is a corresponding element of the same object-type in the second structure. This is a conservatively mixing type of operation.

Internal crossover: Randomly select an element of the first structure. Find an element of the same type in the second If there is not one, then exit with no change. structure. Otherwise, perform a noncomposite crossover between these two elements and replace the original elements with the new This provides a less conservative mixing, but a more ones. conservative method than mutation of avoiding local optima. **Delete-Insert:** Randomly select an element of the first structure. If there is an element of the same type in the second structure, remove the element from the first structure and replace the same type element from the second structure with the first structure's element. Otherwise, delete the element from the first structure and add it to the second structure. If the donor structure becomes empty because of this removal, generate a new one-element classifier of the same object-type that was removed to keep the population at the same level. This operation is designed to generate composite classifiers with a variable number of composition elements.

For each pair of structures, one of these three composite crossing-over operations is selected each generation.

**Performance Measures:** "Performance" is simply how well the structure being evaluated fulfills the objective(s) of the task domain. Three types of performance were used in this model: <u>Pl:</u> Pl is based upon the M3 match algorithm of (Booker, 1985). Adapted to a binary alphabet, Pl is equal to the length (L) of the structure for a perfect match, and  $(m/L^2)$  for a nonperfect match, where m is the number of matching positions. <u>P2:</u> P2 is an adaptation of M3 for composite structures. The Pl of each element in the composite structure is calculated against each entry on the interface board; the maxima for each element are summed; and this value is returned as the performance of that classifier. P2, unlike P3, does not take into account matching the number of elements against the number of environment pattern components: these are unknown to the system. It was assumed that

a classifier with extraneous elements would receive little or no competitive advantage over one with highly matching, nonextraneous elements since only one element of each object-type is allowed in the composite classifier. As we elaborate under "Results and Discussion", this assumption may be invalid. <u>P3:</u> P3 provides an estimation of "offline" performance and was the metric for comparison of performance among the models. "Offline performance" is a term coined by (DeJong, 1975) to indicate the maximum performance score present in a population at a given time-step. In contrast, "online performance" (DeJong, 1975) refers to the average performance score of an entire population at a given time-step. We chose to use a metric of offline performance to indicate the best information that could be expected at a given time-step with each of the models. P3 is calculated as follows:

<b>P3</b>	=				m	 	where	m	=	number	of	matching	bi	ts
		n	(1	+	k	n )				across	all	L element	s;	
			• •		-			n	=	number	of	objects	in	the
										enviro	nmer	nt;		
								k	=	number	of	elements	in	the

classifier;

The factor "m" reflects point-by-point accuracy. Division by (1 + |k - n|) penalizes classifiers whose number of elements differs from the actual number of objects in the environment. Division by n normalizes all results across variable environments. P3 thus represents a proportional accuracy. The maximum value of P3 is 21, implying a complete point-by-point match and an equivalent number of elements in the event classifier as there are objects in the environment. No attempt was made in this set of experiments to determine if object types matched: it was assumed that good attribute matches implied easy object identification.

#### Experimental Groups:

<u>C1:</u> Performance (P3) with no genetic algorithms invoked. This group represented expected results on the basis of chance and served as a control. <u>C2:</u> Performance (P3) with genetic algorithms invoked in the message classifiers but not the event classifiers. Event classifiers were constructed from messages posted to the message to event classifiers interface board in the following manner: A random selection of object types was made to indicate the number and kinds of elements in the event classifier. If there were one or more messages of this type on the interface board, one of these was selected at random to be an element in a classifier. If not, a message was selected at random and transformed into this type. Fifty event classifiers were constructed in this manner at the end of the genetic generation cycle, and the maximum P3 value was cal-

culated. This group was used as a secondary control to determine the efficacy of the composite genetic operators and the P2 measure. The \*generation-limit\* was set at 200.

<u>C3:</u> Performance (P3) with genetic algorithms invoked in the event classifiers and not the message classifiers. Each cycle the message classifiers with the highest P1 values were posted to the interface board. Again, this group served as a secondary control to determine how the composite genetic operators and the P2 measure affected performance. The \*generation-limit\* was set at 200. <u>T1:</u> Performance (P3) with genetic algorithms invoked at the event classifier level at half the rate of the message classifier level. The event classifiers underwent 200 generations interleaved into 400 generations of the message classifiers.

<u>T2:</u> Performance (P3) with genetic algorithms invoked at the message classifier level at half the rate of the event classifiers. The message classifiers underwent 200 generations interleaved into 400 generations of the event classifiers.

<u>T3:</u> Performance (P3) with genetic algorithms invoked at an equal rate at the message classifier and event classifier levels. The \*generation-limit\* was set at 200.

Ten runs for each group were conducted under noise levels of 0.25 (high), 0.125 (medium), and 0.0625 (low).

Statistical Methods: Analysis of variance (ANOVA) was used to test for differences between group means at each noise level ( $\dot{A} = 0.05$ ). The null hypothesis tested was: Cl = Tl = C2 = T2 = C3 = T3 (each value represents a group P3 mean). Duncan's multiple range test was used to discriminate between groups when significant differences were found. All analyses were performed on the Statistical Analysis System (Barr et al., 1979).

### IV RESULTS AND DISCUSSION

Results: The experimental results are illustrated in Tables IV-1, IV-2 and IV-3, for the low, medium, and high noise tests respectively. The following observations are significant:

<u>Result 1:</u> In all noise levels, C2, the group in which genetic algorithms were invoked at the message classifier level but not the event classifier level, performed significantly better than any of the other models. In both the low and medium noise environments, C2 obtained perfect scores of 21 on two trials. <u>Result 2:</u> An ANOVA of C2 performance means across all noise levels indicated no significant difference in the performance of C2 in low, medium, and high noise environments. This result is consistent with previous findings (Part 1: Test 2 in Pettit and Swigger, 1983).





MEAN PERFORMANCE (P3)

<u>Result 3:</u> In the low and medium noise tests, T1, in which the event classifiers underwent adaptation at half the rate of the message classifiers, performed significantly better than T3, in which genetic algorithms were invoked at equal rates and for the same number of iterations for each level. <u>Result 4:</u> In the low and medium noise tests, T1 (event classifiers at rate, message classifiers at 400 generations) and T2 (event classifiers at twice rate, message classifiers at 200 generations) occupied the same equivalence class. <u>Result 5:</u> T1 scored consistently above all the other models except C2. However, at a sample size of 10, the overlap between classes made this relative ranking statistically insignificant.

Discussion: The above observations suggest the following conclusions:

The observations that C2 performed significantly better than any of the other models (Result 1) and that the noise level had no significant effect on its performance (Result 2) suggest that genetic algorithms as employed in the noncomposite case are highly robust in promoting noise filtering of message content. Further experimentation should consider determining the minimum number of iterations necessary for adequate convergence to reduce tracking time.

On the other hand, the models utilizing the composite genetic operators and performance measure P2 did not perform significantly better than the control Cl. Results 3 and 4, combined with a trend suggested by Result 5, suggest that some factor(s) in the composite models is(are) actually disrupting performance. In the following analysis we seek to isolate that factor.

As explored in (Holland, 1975), (DeJong, 1975), (Booker, 1985) and others, the following factors present in our models affect the performance of genetic models operating without composition in nonstochastic environments: population size, mutation rate, crossover rate, selection metrics (P1 and P2 in our models), and number of iterations (\*generation-limit\* in our models). The first three factors were held constant across all models and can therefore be disregarded.

Increasing the number of iterations (\*generation-limit\*) usually improves the performance of genetic models due to their underlying convergent mechanism. However, the observation (Result 4) that T1 and T2, with a 200% difference in \*generation-limit\*s, were not significantly different in even the low noise environment suggests that the problem factor(s) damp the effect of increased iterations. The two most likely sources of such an effect are the selection metric (P2) and the composite operators themselves. A highly inefficient or directionally inaccurate selection pressure

would slow or impair completely the convergence mechanism. The use of composite operators inappropriate for efficiently searching the feature hyperspace would produce similar difficulties.

Result 5, although resting on tenuous statistical ground, at least weakly suggests that the higher quality of information presented by increased iterations of the message classifiers serves to offset somewhat the negative influence of the problem factor(s). This would indicate that the richness of information provided by the message classifiers (Results 1 and 2) is not being severely disrupted in the composite model. We combine this weak indicator with the statistically strong observations that: 1) in low and medium noise environments, T1 (best information quality and 200 iterations) and T2 (less information quality and 400 iterations) were not different; 2) T2 (400 iterations) and T3 (200 iterations and same information quality as T2) were not different; and 3) T1 and T3 with the same number of iterations but different informa-One would expect that if the comtion quality were different. posite operators were inappropriate, information would be reshuffled to such a degree that the integrity of the search process would be lost, and T1 would never be statistically better than T3 (in contrast to Result 3). However, if the selection metric were directionally appropriate but seriously inefficient, the richness of the information would not be randomly disrupted, but the contribution of "good" relative to "meaningless" information in succeeding generations would be severely reduced. Under these circumstances, the richer the initial information the better the performance over the same number of iterations.

From this analysis, we suggest that the use of an attribute-byattribute matching metric (as P2 and Booker's M3 are) is insufficient for composite genetic models. There is a need for a metric which can capture the underlying structure of the "events" dynamically, especially when prior information about this structure is unavailable. We are currently investigating the use of clustering techniques, including conceptual clustering, to devise this metric.

Theoretically, the composite genetic operators do not violate the assumptions of the genetic algorithms. However, since the performance of the genetic operators is primarily affected by the quality of "reproductive information" provided in the selection process we will not make conclusions concerning their empirical validity from these data.

### V CONCLUSION

This experiment examined the use of composite genetic operators and a composite adaptation of a noncomposite selection metric in a bi-level, genetic algorithm-based system for message classification into events in low, medium and high noise environments. Our underlying motivation is the eventual construction of a system for interpreting asynchronous, noisy message traffic in situation assessment problems where these interpretations (events) cannot be predefined but must be acquired during the operation of the system. To this end, in this experiment we have determined that 1) genetic algorithms are highly robust in filtering even random noise from message traffic; and 2) the attribute matching selection metric used extensively in genetic algorithm-based systems is insufficient for a composite model. Both of these are critical issues in the construction of such a system.

Our plans for future research in this area include: 1) devising of a suitable composite selection metric based on clustering theory; 2) examination of the effects of number of iterations on convergence optimization; 3) incorporation of multi-objective functions for optimization, such as object identification and location; 4) introduction of a structured representation similar to (Forrest, 1985)'s use of semantic networks; 5) examination of the online performance of our model in tracking objects which gradually come and go in the environment; and 6) recognition by our model of multiple instances of the same kind of object.

#### ACKNOWLEDGMENTS

This research was supported in part under Merit Technology's internal research and development program in machine learning and in part by the graduate research program in computer science at North Texas State University. We would like to thank Dr. Kathleen Swigger of NTSU for her generous support, and also Gregg Jernigan and his artificial intelligence staff at Merit for their encouragement and suggestions.

#### REFERENCES

Barr, A.J., Goodnight, J.H., Sall, J.P., Blair, W.H., and Chilko, D.M. (1979) <u>SAS User's Guide</u>. SAS Institute. Raleigh, N. Carolina.

Bethke, A.D. (1980) <u>Genetic Algorithms as Function Optimizers</u>. PhD dissertation. Ann Arbor: University of Michigan.

Booker, L. B. (1985) Improving the performance of genetic algorithms in classifier systems. In <u>Proceedings of an International</u> <u>Conference on Genetic Algorithms and their Applications</u>. J.J. Grefenstette (Ed.) pp. 80-92. Pittsburgh:Carnegie-Mellon University.

Brindle, A. (1981) <u>Genetic Algorithms for Function Optimization</u>. Ph.D. dissertation, University of Alberta.

Carson, H.L. (1975) The genetics of speciation at the diploid level. <u>The American Naturalist</u>, Vol. 109, No. 965, pp. 83-92.

Chapman, R.W., Avise, J.C., and Asmussen, M.A. (1979) Character space restrictions and boundary conditions in the evolution of multistate characters. <u>Journal of Theoretical Biology</u>, 80, pp. 51-64.

DeJong, K.A. (1975) <u>An Analysis of the Behavior of a Class of Ge-</u> netic <u>Adaptive Systems</u>. PhD dissertation. Ann Arbor: University of Michigan.

Forrest, S. (1985) Implementing semantic network structures using the classifier system. In <u>Proceedings of An International Confer-</u> <u>ence on Genetic Algorithms and their Applications</u>. J.J. Grefenstette (Ed.). pp. 24-44. Pittsburgh: Carnegie-Mellon University.

Goldberg, D. E. (1985) Genetic algorithms and rule learning in dynamic system control. In <u>Proceedings of An International Confer-</u> ence on Genetic Algorithms and their Applications. J.J. Grefenstette (Ed.). pp. 8-15. Pittsburgh: Carnegie-Mellon University.

Grefenstette, J.J., Gopal, R., Rosmaita, B., and Van Gucht, D. (1985) Genetic algorithms for the traveling salesman problem. In Proceedings of An International Conference on Genetic Algorithms and their Applications. J.J. Grefenstette (Ed.). pp. 160-168. Pittsburgh: Carnegie-Mellon University.

Holland, J. H. (1975) <u>Adaptation in Natural and Artificial</u> <u>Systems.</u> Ann Arbor:University of Michigan Press.

Holland, J. H. (1986) Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In <u>Machine Learning: An Artificial Intelligence Approach</u> <u>Volume II</u>. R.S. Michalski, J. Carbonell, and T. Mitchell (Eds.). pp. 625-646. Los Altos:Morgan Kaufman Publishers.

Michalski, R. S. (1983) A theory and methodology of inductive learning. In <u>Machine Learning: An Artificial Intelligence Approach</u> <u>Vol. I.</u> R.S. Michalski, J. Carbonell, and T. Mitchell (Eds.). pp. 83-134. Palo Alto:Tioga Publishing Co.(now Morgan Kaufmann).

Michalski, R.S. and Stepp, R.E. (1983) Learning from observation: conceptual clustering. In <u>Machine Learning: An Artificial Intel-</u> <u>ligence Approach Vol. I</u>. R.S. Michalski, J. Carbonell, and T. Mitchell (Eds.). pp. 331-364. Palo Alto:Tioga Publishing Co.(now Morgan Kaufmann).

Michalski, R.S. (1986) Understanding the nature of learning. In <u>Machine Learning: An Artificial Intelligence Approach Volume II</u>. R.S. Michalski, J. Carbonell, and T. Mitchell (Eds.). pp. 3-26. Los Altos:Morgan Kaufman Publishers.

Pettit, E. J. and Swigger, K. (1983) An analysis of genetic-based pattern tracking and cognitive-based component tracking models of adaptation. In <u>Proceedings of the National Conference on Artificial Intelligence</u>, (Washington, D. C.) Los Altos:William Kaufmann.

Quinlan, J.R. (1986) The effect of noise on concept learning. In <u>Machine Learning: An Artificial Intelligence Approach Volume II</u>. R.S. Michalski, J. Carbonell, and T. Mitchell (Eds.). pp. 149-166. Los Altos:Morgan Kaufman Publishers.

Quirin, W.L. (1978) <u>Probability and Statistics</u>. New York:Harper & Row.

Schlimmer, J.C., & Fisher, D. (1986) A case study of incremental concept induction. In <u>Proceedings of the Fifth National Conference on Artificial Intelligence.</u> Vol 1. pp. 496-501. Los Altos: Morgan Kaufmann.

Schlimmer, J.C., & Granger, R.H., Jr. (1986) Beyond incremental processing: tracking concept drift. In <u>Proceedings of the Fifth</u> <u>National Conference on Artificial Estelligence.</u> Vol 1. pp. 502-507. Los Altos: Morgan Kaufmann.

Smith, S.F. (1980) <u>A Learning System Based on Genetic Adaptive</u> <u>Algorithms</u>. PhD dissertation. Ann Arbor: University of Michigan.

Waterman, D. (1975) Adaptive production systems. In <u>Proc. 4th In-</u> ternational Joint Conference on Artificial Intelligence. Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# Simplifying Decision Trees

J. R. Quinlan<sup>1</sup> Artificial Intelligence Laboratory Massachusetts Institute of Technology 545 Technology Square Cambridge, MA 02139

Abstract: Many systems have been developed for constructing decision trees from collections of examples. Although the decision trees generated by these methods are accurate and efficient, they often suffer the disadvantage of excessive complexity and are therefore incomprehensible to experts. It is questionable whether opaque structures of this kind can be described as knowledge, no matter how well they function. This paper discusses techniques for simplifying decision trees while retaining their accuracy. Four methods are described, illustrated, and compared on a test-bed of decision trees from a variety of domains.

# 1. Introduction

Since people began building knowledge-based systems, it has become painfully obvious that the ability to function at an expert level in some task domain does not necessarily confer a corresponding ability to articulate this know-how. The knowledge for most early and many current expert systems has been amassed by an interview process in which a *knowledge engineer* interacts with a domain expert to extract and refine a set of *rules*. This process can be taxing for all concerned because the expert, as Waterman (1986) puts it,

"has a tendency to state [his] conclusions and the reasoning behind them in general terms that are too broad for effective machine analysis ... the pieces of basic knowledge are assumed and combined so quickly that it is difficult for him to describe the process."

Consequently, the productivity of the interview method is usually low. This led Feigenbaum (1981) to identify knowledge acquisition as the "bottleneck" problem in building knowledge-based systems.

One way around this bottleneck, long advocated by Donald Michie (1983) and others, uses inductive methods to extract general rules from concrete examples. The expert is not asked

<sup>1</sup>Permanent address: School of Computing Sciences, New South Wales Institute of Technology, Sydney, N.S.W. 2007, Australia.

to articulate his skill but instead to provide a framework of important concepts in the task domain, augmented perhaps by a collection of tutorial examples; the hard work is carried out by a suitable induction engine. Most researchers in Machine Learning will be familiar with Meta-DENDRAL and its synthesis of chemical knowledge (Buchanan and Mitchell, 1978) and with AQ11's results on soybean diagnosis (Michalski and Chilausky, 1980). The feasibility of this inductive approach to knowledge acquisition has also been confirmed in several industrial projects, such as British Petroleum's recent successful construction of a 2500-rule expert system for the design of hydrocarbon separation vessels in just one man-year (*Expert Systems User*, August 1986, pp16-19).

Many current commercial induction packages (including *Ex-Tran*, *RuleMaster* and *1st-Class*) express the derived rules in the form of *decision trees*. From the standpoint of execution efficiency this is a simple and economical representation, but the trees can become complex and thus opaque (Michie, 1986). If a decision tree that measures up very well on the performance criterion is nevertheless totally incomprehensible to a human expert, can it be described as *knowledge*? Under the common-sense definition of this term as material that might be assimilated and used by human beings, it is not, in just the same way that a large program coded in assembly language is not knowledge.

This paper examines four methods for improving the intelligibility of decision trees and thereby making them more knowledge-like. Three of the methods involve *pruning* the decision tree by replacing one or more subtrees with leaves, while the remaining method reformulates the decision tree as a set of production rules. Section 2 introduces the methods and illustrates their operation with respect to a small but real example. Section 3 presents an empirical comparison of the methods using sets of decision trees from six task domains.

# 2. Methods for Simplifying Decision Trees

Induction algorithms that develop decision trees view the task domain as one of *classification*. The underlying framework consists of a collection of *attributes* or properties which are used to describe individual *cases*, each case belonging to exactly one of a set of *classes*. Attributes may be either continuous or discrete. A case's value of a continuous attribute is always a real number while its value of a discrete attribute is one of a small set of possible values for that attribute. In real-life tasks it is also important to recognise that a case may have *unknown* values for one or more of the attributes.

A decision tree may be either a leaf identified by a class name, or a structure of the form

$$\begin{array}{ccc} C_1 \colon & D_1 \\ C_2 \colon & D_2 \\ & \ddots & \ddots \\ C_n \colon & D_n \end{array}$$

where the  $C_i$ 's are mutually exclusive and exhaustive logical conditions and the  $D_i$ 's are themselves decision trees. The set of conditions involves only one of the attributes, each condition being

$$A < T$$
 or  $A > T$ 

for a continuous attribute A, where T is some threshold, or

$$A = V$$
 or  $A$  in  $\{V_i\}$ 

for a discrete attribute A, where V is one of its possible values and  $\{V_i\}$  is a subset of them. To improve legibility, the non-leaf subordinate decision trees above will be indented when the trees are printed.

Such a decision tree is used to classify a case as follows. If the tree is a leaf, we simply determine the case's class to be the one nominated by the leaf. If the tree is a structure, we find the single condition  $C_i$  that holds for this case and continue with the associated decision tree. The only complexity arises when the value of the attribute appearing in the  $C_i$ 's is unknown. In this eventuality we explore all the decision trees associated with the structure and combine their findings with weights proportional to the estimated probability of the associated condition being satisfied. (Quinlan, 1986) discusses the procedure in more detail.

Figure 1 shows such a decision tree for the diagnosis of hypothyroid conditions with classes  $\{primary\ hypothyroid,\ secondary\ hypothyroid,\ compensated\ hypothyroid,\ negative\}$ . Some attributes such as TSH and FTI are continuous and have real values, while attributes like thyroid surgery, with possible values  $\{t, f\}$ , are discrete. To classify a case with this tree, we would first enquire whether the value of TSH was greater than 6.05. If the value was below this threshold we would continue with the decision tree commencing with T4U measured = t, while a value above this threshold would lead us to the decision tree headed FTI < 64.5. In either case we would continue in similar fashion until a leaf was encountered.

The set of cases with known classes from which a decision tree is induced is called the *training set*. Other collections of cases not seen while the tree was being developed are known as *test sets* and are commonly used to evaluate the performance of the tree.

This paper focusses on simplifying decision trees, not with the inductive methods used to construct them in the first place. Various ways of developing trees from training sets may be found in (Breiman, Friedman, Olshen and Stone, 1984), (Kononenko, Bratko and Roškar, 1984) and (Quinlan, 1986).

```
TSH < 6.05:
    T4U measured = t: negative (1918)
    T4U measured = f:
        age > 43.5: negative (58)
        age < 43.5:
            query hypothyroid = f: negative (41)
    1
            query hypothyroid = t: secondary hypothyroid (1)
        TSH > 6.05:
   FTI < 64.5:
        thyroid surgery = f:
    L
        I
            T3 < 2.3: primary hypothyroid (51)
            T3 > 2.3:
    L
                sex = M: negative (1)
            T
            1
                sex = F: primary hypothyroid (4)
        thyroid surgery = t:
            referral source = SVI: primary hypothyroid (1)
        I
            referral source = <other>: negative (2)
   FTI > 64.5:
        on thyroxine = t: negative (32)
        on thyroxine = f:
            thyroid surgery = t: negative (3)
            thyroid surgery = f:
Î
                TT4 < 150.5: compensated hypothyroid (120)
            TT4 > 150.5: negative (6)
```

Figure 1: Sample Decision Tree

## 2.1 Cost-Complexity Pruning

Breiman et al (1984) describe a two-stage process in which a sequence of trees  $T_0, T_1, ..., T_k$ is generated.  $T_0$  is the original decision tree and each  $T_{i+1}$  is obtained by replacing one or more subtrees of  $T_i$  with leaves until the final tree  $T_k$  is just a leaf. The second stage evaluates these trees and selects one of them as the final pruned tree.

Consider a decision tree T used to classify each of the N cases in the training set from which T was generated, and let E of them be misclassified. If L(T) is the number of leaves in T, Breiman *et al* define the *cost-complexity* of T as the sum

$$rac{E}{N} + lpha imes L(T)$$

for some parameter  $\alpha$ . Now, suppose we were to replace some subtree S of T by the best possible leaf. In general, the new tree would misclassify M more of the cases in the training set but would contain L(S) - 1 fewer leaves. This new tree would have the same

```
36-3
```

cost-complexity as T if

$$\alpha = \frac{M}{N \times (L(S) - 1)}$$

As before, let  $T_0$  be the original tree. To produce  $T_{i+1}$  from  $T_i$  we examine each non-leaf subtree of  $T_i$  to find the minimum value of  $\alpha$  above. The one or more subtrees with that value of  $\alpha$  are then replaced by their respective best leaves.

To illustrate the process, consider the decision tree of Figure 1. This was generated from 2514 cases, where the number in parentheses after each leaf shows how many of these cases are covered by that leaf.<sup>2</sup> Consider the subtree

T4U measured = t: negative (1918)
T4U measured = f:
| age > 43.5: negative (58)
| age < 43.5:
| | query hypothyroid = f: negative (41)
| | query hypothyroid = t: secondary hypothyroid (1)</pre>

The vast majority of cases at the leaves of this subtree are of class *negative* which is clearly the best leaf. If the subtree were replaced by the leaf *negative* the new tree would misclassify the lone non-*negative* case, so M is 1. The new tree would also have three fewer leaves, giving a value for  $\alpha$  of 0.00013 at which the cost-complexity of the original and modified trees would be equal. This is the lowest such value for any subtree, so the tree  $T_1$  would be formed by replacing this subtree as above.

The second stage of this process abandons the cost-complexity model and attempts to select one of the  $T_i$ 's on the basis of reliability alone. We cannot assess this simply from the proportion of cases in the original training set that are misclassified. Whatever induction algorithm was employed has almost certainly built the original tree to fit the training set and thus the error rate on these cases would be expected to understate the error rate on unseen cases. We therefore assume some test set containing N' cases and use each  $T_i$  to classify all of them. Let E' be the minimum number of errors observed with any  $T_i$ , with the standard error of E' being given by

$$se(E') = \sqrt{rac{E' imes (N' - E')}{N'}}$$

The tree selected is the smallest  $T_i$  whose observed number of errors on the test set does not exceed E' + se(E').

<sup>&</sup>lt;sup>2</sup>The counts do not sum to 2514 because cases with unknown values of tested attributes cannot be associated with any one leaf and are therefore not included.

```
TSH < 6.05: negative (2018)
TSH > 6.05:
| FTI < 64.5: primary hypothyroid (62)
| FTI > 64.5:
| | on thyroxine = t: negative (32)
| | on thyroxine = f:
| | | thyroid surgery = t: negative (3)
| | thyroid surgery = f:
| | | TT4 < 150.5: compensated hypothyroid (120)
| | | | TT4 > 150.5: negative (6)
```

Figure 2: Decision Tree After Cost-Complexity Pruning

In this example, a test set containing 629 cases gave a sequence of eight trees,  $T_0$  being the original tree and  $T_7$  the leaf *negative*. The selected tree was  $T_4$  which appears in Figure 2. This tree is indeed a great deal simpler than the original and would qualify as 'knowledge' under the most stringent criterion. Notice that the class *secondary hypothyroid*, which is represented by just a single case in the training set, has sensibly been omitted.

Nevertheless, cost-complexity pruning raises several problematic issues. First, it is unclear why the particular cost-complexity model used above is superior to other possible models such as the product of error rate and number of leaves. Secondly, it seems anomalous that the cost-complexity model used to generate the sequence of subtrees is abandoned when the best tree is selected. Finally, the procedure requires a test set distinct from the original training set; the authors show, however, that a cross-validation scheme can be employed to generate these estimates at the time the original tree is constructed, but at the expense of a substantial increase in computation.

## 2.2 Reduced Error Pruning

Rather than form a sequence of trees and then select one of them, a more direct procedure suggests itself as follows. We again assume a separate test set, each case in which is classified by the original tree. For every non-leaf subtree S of T we examine the change in misclassifications over the test set that would occur if S were replaced by the best possible leaf. If the new tree would give an equal or fewer number of errors and S contains no subtree with the same property, S is replaced by the leaf. The process continues until any further replacements would increase the number of errors over the test set.

Using the same example of Figure 1 and the same test set as before, reduced error pruning generates the tree shown in Figure 3.

```
TSH < 6.05: negative (2018)
TSH > 6.05:
    FTI < 64.5:
        thyroid surgery = f: primary hypothyroid (59)
    1
        thyroid surgery = t:
            referral source = SVI: primary hypothyroid (1)
    L
            referral source = <other>: negative (2)
        Ł
    FTI > 64.5:
I
        on thyroxine = t: negative (32)
1
    T
        on thyroxine = f:
            thyroid surgery = t: negative (3)
    L
            thyroid surgery = f:
    Î
I
                TT4 < 150.5: compensated hypothyroid (120)
1
            L
Î
    I
        1
                TT4 > 150.5: negative (6)
            1
```

Figure 3: Decision Tree After Reduced Error Pruning

As with cost-complexity pruning, this process generates a sequence of trees. Its rationale is clearer, though, since the final tree is the most accurate subtree of the original tree with respect to the test set and is the smallest tree with that accuracy. The disadvantages of the method are, first, that it again requires a separate test set and second, that parts of the original tree corresponding to rarer special cases not represented in the test set may be excised.

## 2.3 Pessimistic Pruning

When the original tree T is used to classify the N cases in the training set from which it was generated, let some leaf account for K of these cases with J of them misclassified. As observed before, the ratio J/K does not provide a reliable estimate of the error rate of that leaf when unseen cases are classified, since the tree has been tailored to the training set. A more realistic error rate might be obtained using the continuity correction for the binomial distribution (Snedecor and Cochran, 1980, pp. 117ff) in which J is replaced by  $J + 1/2.^3$ 

Let S be a subtree of T containing L(S) leaves and let  $\Sigma J$  and  $\Sigma K$  be the corresponding sums over the leaves of S. A more pessimistic view of S is that it will misclassify  $\Sigma J + L(S)/2$  out of  $\Sigma K$  unseen cases, where the standard error of this number of misclassifications can be determined as before. If S were replaced by the best leaf, let E be the number of cases from the training set that it misclassifies. The pessimistic pruning method replaces S by the best leaf whenever E + 1/2 is within one standard error of  $\Sigma J + L(S)/2$ .

<sup>&</sup>lt;sup>3</sup>This makes the unsurprising assumption that J/K < 0.5.

All non-leaf subtrees are examined just once to see whether they should be pruned but, of course, sub-subtrees of pruned subtrees need not be examined at all.

To illustrate the idea we return to the subtree of Figure 1 that commences with the condition T4U measured = t. As before,  $\Sigma K$  is 2018, L(S) is 4,  $\Sigma J$  is 0, so the estimate of the number of errors due to S is 2.0 with standard error 1.41. If the subtree is replaced by the leaf negative it will give one error, so E is 1. Since 1+1/2 < 2.0+1.41, pessimistic pruning would indeed replace this subtree. Repeating this evaluation on all subtrees of T gives a pruned tree identical to that of Figure 2.

This method has two advantages. It is much faster than either of the preceding methods since each subtree of T is examined at most once. Unlike these methods, it does not require a test set separate from the cases in the training set from which the tree was constructed.

# 2.4 Simplifying to Production Rules

This form of simplification does not give a smaller decision tree at all but instead develops an 'equivalent' set of *production rules*, a representation medium widely used in expert systems (Winston, 1984). The process has two stages: individual production rules are first generated and polished, and then the rules produced are evaluated as a collection.

Whenever a decision tree is used to classify a case, a path is established between the top of the tree and one of its leaves. In order for the case to reach that leaf, it must have satisfied all the conditions along the path. For example, any case that is classified as *negative* by the last leaf of the decision tree in Figure 1 must satisfy all the conditions

$$TSH > 6.05$$
  
 $FTI > 64.5$   
on thyroxine = f  
thyroid surgery = f  
 $TT4 > 150.5$ 

Every leaf of a decision tree thus corresponds to a production rule of the form

if 
$$X_1 \wedge X_2 \wedge \ldots \wedge X_n$$
 then class c

where the  $X_i$ 's are conditions as before and c is the class of the leaf.

Merely rewriting a tree as the collection of these equivalent production rules would not represent any simplification at all. Instead, the first stage examines each production rule to see whether it should be generalised by dropping conditions from its left-hand side. Let  $X_i$  be one of the conditions and consider those cases in the training set that satisfy all the other conditions in the rule. With respect only to these cases, the relevance of  $X_i$  to determining whether a case belongs to class c (given that the other conditions are satisfied) can be summarised by the  $2 \times 2$  contingency table

		not
	class c	class c
satisfies $X_i$	sc	sī
does not satisfy $X_i$	<u>s</u> c	<u>sc</u>

where sc is the number of these cases that satisfy  $X_i$  and belong to class  $c, s\overline{c}$  is the number that satisfy  $X_i$  but belong to some class other than c, and so on. Fisher's exact test (Finney, Latscha, Bennett and Hsu, 1963) can then be invoked to assess the probability that the division by  $X_i$  arises merely from chance or, in other words, the significance level at which we can reject the hypothesis that  $X_i$  is irrelevant to whether a case belongs to class c.<sup>4</sup> Each  $X_i$  is examined in turn to find the one that has the least relevance to classification and, unless the hypothesis that this  $X_i$  is not significant can be rejected at the 0.1% level or better, the condition is discarded and the process repeated.

Consider the rule above. When the training cases that satisfy all conditions other than the first are examined, the table for the condition TSH > 6.05 comes out to be

		not
	class	class
	negative	negative
TSH > 6.05	6	0
TSH < 6.05	154	0

which shows that this condition is entirely irrelevant. On the other hand, the table of cases satisfying all conditions other than the last is

		not
	class	class
	negative	negative
TT4 > 150.5	6	0
TT4 < 150.5	0	120

which is significant at better than the 0.1% level. Repeated application of the above process reduces the original rule to one with a single condition

if TT4 > 150.5 then class negative

The final step in this first stage is to estimate a *certainty factor* for the simplified rule, using a device similar to that of pessimistic pruning. If the left-hand side of a rule is

<sup>&</sup>lt;sup>4</sup>I am indebted to Donald Michie of the Turing Institute for making me aware of this test and its advantages over the approximate  $\chi^2$  test.

satisfied by V cases in the training set, W of which belong to the class indicated by the right-hand side, the certainty factor of the production rule is taken as (W - 1/2)/V. In the example above, the training set contains 246 cases that match the left-hand side, all of them being class *negative*, so this rule's CF is 99.8%.

Note that we need not develop one rule for each leaf of the decision tree. Some leaves give rise to identical rules while other leaves generate vacuous rules from which all conditions have been dropped. The number of rules is generally smaller than the number of leaves.

The second stage of this process looks how well the rules will function as a set. This evaluation depends on the way in which the rules will be used. A simple strategy has been adopted here:

To classify a case, find a rule that applies to it. If there is more than one, choose the rule with the higher certainty factor. If no rule applies, take the class by default to be the most frequent class in the training set.

For each rule in turn, we now determine how the remaining rules would perform on the training set if this rule were omitted. If there are rules whose omission would not lead to an increased number of errors classifying the cases in the training set, or would even reduce it, the least useful such rule is discarded and the process repeated.

Continuing the example, the decision tree of Figure 1 is reduced by this method to just three rules:

if	TSH < 6.05	then	class negative [99.9%]
if	thyroid surgery = $f \land TSH > 6.05 \land FTI < 64.5$	then	class primary hypothyroid [97.5%]
if	on thyroxine = $f \land$ thyroid surgery = $f \land$ $TSH > 6.05 \land$ $TT4 < 150.5 \land$ FTI > 64.5	then	class compensated hypothyroid [99.6%]

As with pessimistic pruning, this method does not require a set of test cases apart from the original training set. In its current implementation it is the slowest of the four treesimplifying methods. The method should be able to be improved by adopting a more sophisticated condition-elimination strategy than the simple hill-climbing approach used above, and by employing a better production rule interpreter.
# 2.5 Other Methods

The four methods of simplifying decision trees certainly do not exhaust all possibilities. The cross-validation method of Breiman *et al* (1984) has already been mentioned. Kononenko *et al* (1984) present an information-based heuristic used in their ASSISTANT system, but this is now being changed to another form of cross-validation (Lavrač, Mozetič and Kononenko, 1986). I have previously experimented with a form of pruning based on the path lengths in the decision tree and observed error rates (Quinlan, Compton, Horn and Lazarus, 1986).

# 3. Empirical Comparison

The performance of a simplification method can be assessed in terms of the clarity and accuracy of its final product. Ideally, the pruned decision tree or set of production rules should be much more comprehensible than the original decision tree but should not be significantly less accurate when classifying unseen cases.

To test how well the methods of the previous section measure up to these two criteria, they were compared using decision trees developed for six task domains. For each domain, the available data was shuffled, then divided into a training set containing approximately two-thirds of the data and two equal-sized test sets. This division was carried out so as to make the proportion of cases belonging to each class as even as possible across the three sets. The training set was used to induce ten decision trees for the domain. Each simplification method was applied to each tree and the resulting classifier evaluated on both test sets.

The six domains include both real-world tasks and synthetic tasks constructed to provide some particular challenge. They are

- Diagnosis of hypothyroid conditions (*Hypothyroid*): This domain has been encountered in the running example of the previous section. The data comes from the archives of the Garvan Institute of Medical Research, Sydney, and covers all 3772 thyroid assays carried out by Garvan's clinical laboratory between January and November 1985. The data uses seven continuous and sixteen discrete attributes with quite high rates of missing information values of four of the attributes are unknown in more than 10% of the cases. The 3772 cases, each belonging to one of four classes, were split into a training set of 2514 and two test sets of 629. This domain is a good starting point because it uses 'live' data from which, warts and all, extremely accurate classifiers can be constructed.
- Discordant assay results (*Discordant*): This domain is taken from the same Garvan data, this time looking to detect anomalous combinations of thyroid hormone values.

There are two classes and the 3772 cases were divided as above. The percentage of discordant cases is very low (about 1.5%) and, in contrast with the first domain, the decision trees generated from this training set perform comparatively poorly on unseen cases.

- Recognising faulty digits (*LED Digits*): The third domain comes from (Breiman *et al*, 1984). Imagine a seven-element representation for a decimal digit such as is commonly found on LED or LCD displays. Each element of a faulty display is subject to a 10% random error, i.e. with probability 0.1 its correct status is inverted. The data consists of 3000 randomly-generated cases, each described in terms of the seven binary attributes, with ten equiprobable classes. The training set contains 2000 cases, the test sets 500 each. This artificial domain is interesting because it tests the ability of the simplification methods to deal with the complex decision trees commonly obtained from noisy training sets.
- Assessing consumer credit applications (*Credit*): The data for this domain was provided by a large bank. Each case concerns an application for credit card facilities described by 9 discrete and 6 continuous attributes, with two decision classes. The 690 cases making up the data are divided into a training set of 460 and two test sets of 115. Some discrete attributes have large collections of possible values (one of them has 14) resulting in broad, shallow decision trees. This data is also both scanty and noisy, giving decision trees that are extremely complex and not very accurate on unseen cases.
- King and rook versus king and knight (*Endgame*): This domain from a chess endgame seeks to decide whether the rook's side can capture the opposing knight and/or checkmate in 3 ply. Positions are described by 39 binary attributes, with all possible board positions giving rise to 551 distinct cases. This domain models an idealised noise-free environment with no missing information in which the accuracy of the decision tree depends only on the completeness of the training set. Here the training set contains 367 cases, the test sets 92 cases each.
- Probabilistic classification over disjunctions (*Prob-Disg*): The last domain is an artificial one designed to model tasks in which only probabilistic classification is possible and which contains explicit disjunctions. There are ten boolean attributes  $a_0$  through  $a_9$  and the criterion used to generate the data can be expressed as: if  $a_0 \wedge a_1 \wedge a_2$  or  $a_3 \wedge a_4 \wedge a_5$  or  $a_6 \wedge a_7 \wedge a_8$  then the class is Y with probability 0.9, N with probability 0.1; otherwise, the class is N with probability 0.9, Y with probability 0.1. (The remaining attribute  $a_9$  is irrelevant.) Because the class of a case is determined probabilistically, no classification procedure can achieve more than 90% accuracy on this task. Six hundred cases with random values for each attribute were generated and classified as above. Of these, 400 are used as the training set, leaving test sets of 100 cases each.

	Original	Cost-	Reduced	Pessimistic	Production
	Decision	Complexity	Error	Pruning	Rule
	Trees	Pruning	Pruning		Form
Hypothyroid	23.6 nodes	11.4 nodes	14.4 nodes	11.0 nodes	3.0 rules
Discordant	52.4 nodes	11.8 nodes	12.4 nodes	13.6 nodes	1.8 rules
LED Digits	92.2 nodes	45.6 nodes	59.0 nodes	56.0 nodes	15.8 rules
Credit	248.0 nodes	9.7 nodes	26.3 nodes	32.5 nodes	7.8 rules
Endgame	88.8 nodes	51.0 nodes	55.6 nodes	62.6 nodes	11.6 rules
Prob-Disj	190.0 nodes	30.4 nodes	43.0 nodes	42.6 nodes	4.2 rules

Table 1: Average Size Before and After Simplification

The results of these experiments are summarised in the following tables. The effectiveness of the simplification methods in reducing the size of the original decision trees is shown in Table 1, each entry being the average over the ten decision trees in that domain. As a general observation, all the methods achieve significant simplification in all domains. Cost-complexity pruning tends to produce smaller decision trees than either reduced error or pessimistic pruning, especially in the *Credit* domain. While the complexity of decision trees and sets of production rules cannot be compared directly, it would appear that the last method achieves the greatest reduction overall, its advantages being particularly noteworthy in the *Prob-Disj* domain.

The other side of the coin is the effect of simplification on classification accuracy. Table 2 shows the results in each domain of using the ten original decision trees and their simplified counterparts to classify cases in the two test sets, expressed as the average percentage of misclassifications over each set. Perhaps surprisingly, the simplified trees on the whole are of superior or equivalent accuracy to the originals, so pruning has been beneficial on both counts. Note, though, that both the cost-complexity and reduced error methods have "seen" the first training set in performing their respective simplifications. The slight superiority of reduced error pruning, coupled with the fact that cost-complexity pruning produces smaller trees, suggests that the latter may be slightly over-pruning. Despite *not* having seen the first test set, the performance of pessimistic pruning is marginally better than cost-complexity pruning averaged over all domains. Simplification to production rules, though, scores pretty clear wins in the last two domains. In the *Prob-Disj* domain in particular, this can be explained by observing that disjunctive concepts tend to scatter cases from some disjuncts throughout the decision tree. Pruning the tree is unable to re-collect these cases, but simplification of rules can.

	Original	Cost-	Reduced	Pessimistic	Production
	Decision	Complexity	Error	Pruning	Rule
	Trees	Pruning	Pruning		Form
Hypothyroid					
Test 1	0.3%	0.4%	0.3%	0.5%	0.3%
Test 2	0.8%	0.7%	0.8%	0.6%	1.0%
Discordant					
Test 1	1.6%	1.1%	1.0%	1.0%	1.1%
Test 2	2.1%	1.6%	1.7%	1.5%	1.5%
LED Digits			***************************************		
Test 1	30.0%	29.9%	27.8%	28.8%	31.3%
Test 2	27.9%	28.7%	28.0%	27.4%	28.3%
Credit					
Test 1	20.2%	14.4%	12.9%	15.8%	15.2%
Test 2	21.0%	17.1%	17.4%	16.4%	17.8%
Endgame				· ·	
Test 1	11.8%	13.8%	10.0%	13.1%	11.1%
Test 2	10.5%	13.4%	11.6%	12.1%	7.3%
Prob-Disj					
Test 1	17.0%	14.2%	10.1%	14.0%	10.0%
Test 2	18.4%	17.2%	17.8%	15.8%	10.0%

 Table 2: Average Error Rates on Test Sets

One further possibility has been explored. There is no obvious way to merge distinct decision trees, so pruned trees from different originals cannot be combined to form a composite tree that reflects the various strengths of its components. No such limitation applies to the production rule representation, though, because the union of sets of rules is itself a set. This line of thought led to a final experiment in which, for each domain, the rule sets produced from all ten decision trees were amalgamated and the collection winnowed as before. The composite rule set was then used to classify all cases in the test sets. The results in Table 3 show that these composite sets of production rules are both compact and accurate classifying mechanisms, matching or outperforming the best of all other methods on nine of the twelve test sets.

	Number Error Rates		Rates
	of Rules	Test 1	Test 2
Hypothyroid	· 3	0.3%	1.0%
Discordant	2	0.6%	1.4%
LED Digits	23	28.2%	25.8%
Credit	11	13.0%	15.7%
Endgame	12	9.8%	5.4%
Prob-Disj	4	10.0%	10.0%

Table 3: Error Rates of Composite Rule Sets

# 4. Conclusion

The intention of this paper has been to investigate methods for simplifying decision trees without compromising their accuracy. The motivation behind this drive towards simplicity is the desire to turn decision trees into knowledge for use in expert systems.

Four methods have been discussed, all of which managed to achieve significant simplification when put to the test on sets of decision trees from six task domains. This simplification was often coupled with an actual improvement in classification accuracy on unseen cases. Two of the four methods needed a separate set of test cases in order to carry out the simplification and, since these did not perform noticeably better than the remaining two methods, the requirement of additional test data is a weakness. The last method, in which decision trees are reformulated as sets of production rules, has proved especially powerful.

# Acknowledgements

I am grateful to the Garvan Institute of Medical Research for providing access to the thyroid data, and to Les Lazarus and Paul Compton in particular for their help. This work has been supported in part by grants from the Australian Research Grants Scheme and the Westinghouse Corporation.

### References

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. (1984), Classification and Regression Trees, Belmont: Wadsworth.

Buchanan, B.G., and Mitchell, T.M. (1978), Model-directed learning of production rules, in *Pattern Directed Inference Systems*, (D.A. Waterman and F. Hayes-Roth, Eds.), Academic Press.

Feigenbaum, E.A. (1981), Expert systems in the 1980s, in State of the Art Report on Machine Intelligence, (A. Bond, Ed.), Maidenhead: Pergamon-Infotech.

Finney, D.J., Latscha, R., Bennett, B.M. and Hsu, P. (1963), Tables for Testing Significance in a  $2 \times 2$  Contingency Table, Cambridge University Press.

Kononenko, I., Bratko, I., and Roškar, E. (1984), Experiments in automatic learning of medical diagnostic rules, Technical Report, Jozef Stefan Institute, Ljubljana, Yugoslavia.

Lavrač, N., Mozetič, I. and Kononenko, I. (1986), An experimental comparison of two learning programs in three medical domains, *Proceedings of ISSEK Workshop 86*, Turing Institute, Glasgow.

Michalski, R.S. and Chilausky, R.L. (1980), Learning by being told and learning by examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis, *International Journal* of Policy Analysis and Information Systems 4, 2.

Michie, D. (1983), Inductive rule generation in the context of the Fifth Generation, Proceedings of the Second International Machine Learning Workshop, University of Illinois at Urbana-Champaign.

Michie, D. (1986), Current developments in expert systems, Proceedings of the Second Australian Conference on Applications of Expert Systems, Sydney.

Quinlan, J.R. (1986), Induction of decision trees, Machine Learning 1, 1.

Quinlan, J.R., Compton, P.J., Horn, K.A. and Lazarus, L. (1986), Inductive knowledge acquisition: a case study, *Proceedings of the Second Australian Conference on Applications of Expert Systems*, Sydney.

Snedecor, G.W. and Cochran, W.G. (1980), *Statistical Methods* (7th edition), Iowa State University Press.

Waterman, D.A. (1986), A Guide to Expert Systems, Addison-Wesley.

Winston, P.H. (1984), Artificial Intelligence (2nd edition), Addison-Wesley.

# Multiple problem spaces in the knowledge design process

### Alain Rappaport\*

**Neuron Data** 444 High St Palo Alto CA 94301

**The Robotics Institute** C-MU, Pittsburgh PA 15213

Abstract: Designing a knowledge base is viewed as a problem solving task in which the skilled individual's knowledge and behavior must be mapped into the system, preserving the compiled knowledge acquired by experience. The expert's problem space is very large, but its breakdown into three major subspaces allows one to formalize this approach. Selective interfaces and high-level heuristics as well as a flexible knowledge representation not only elicitate knowledge but also allow shifts in the designer's methodologies used in the knowledge design task. High-level programming, stressing the importance of the mental models, should allow to bypass the current bottlenecks of having to decompile the knowledge into a restricted representation and then reconstruct the control structures to recover the expertise. Thus problem solving, or knowledge design, becomes a functionality available to domain experts themselves.

The following reflections aim at the construction of a comprehensive theory of knowledge acquisition and transfer, in the context of a direct relation between the domain-expert and the machine. This work is linked to the development and use of the NEXPERT<sup>M</sup> hybrid knowledge-based system.

#### KNOWLEDGE DESIGN

A problem is defined as an event which requires the immediate allocation of resources because it cannot be dealt with in an automated fashion and/or the event is too new or different. Thus, the dependence between the notions of problem and that of observer of that problem makes this observer's behavior a critical element of the description of problems in general.

The lack of formalization of human knowledge is not solely due to its variety of expressions and to the complexity of the thought processes. It is also due to the fact that a lot of the interesting human knowledge is compiled knowledge, resulting from more or less long experiential learning. It is reflected at the behavioral level by the quick allocation of the correct resources and a high degree of efficiency in the problem-solving activity.

The task of formalizing knowledge of a domain and the associated mental processes in the building of a knowledge base application is itself a problem-solving task, taking the information processing viewpoint. The progressive assembly of chunks of knowledge reflects a task of search through a large problem-space: the expert's domain. We call this task of incrementally discovering and refining knowledge "knowledge design". It can be studied according to the "problem-space" principle (Card et al. 1983). Since the expression of knowledge into the knowledge system reflects the designer's short-term memory at any time, the design of the tool itself should help to correctly focus the designer's attention, optimizing the search function by providing heuristics and elicitating high-level problem-solving techniques.

### DECOMPILATION OR NOT

Human knowledge expresses itself at the behavioral level and not only as textbook knowledge. The behavior level is critical in the nature of expert knowledge, versus novice knowledge. It is of course the case that two people could have the same book knowledge. However, the distinction between the expert and the novice will be in the speed of recognition (simon chess) and problem-solving techniques used when confronted to a problem. The expert has accumulated experience which results in rules of thumb, intuitions and inductive reasoning.

These elements account for a difference in performance. Is expert knowledge only an increase in performance of the search functions? Learning processes are certainly not only limited to skill acquisition. Rather, they result in the acquisition of compiled knowledge, or elements of knowledge which properties have somehow emerged from the assembly and/or transformation of past, more elementary or obsolete descriptions. It is the complex result of linear mechanisms (practice) and, most importantly, non-linear ones (analogy, discovery, examples...). A given element of knowledge, be it a rule or a structured object, embodies both objective textbook knowledge and subjective, proprietary heuristics for its manipulation relative to the others during problem-solving. For the moment, we shall focus on the translation of human knowledge into a machine program with a maximum comparative efficiency.

How much of this human knowledge needs to be decompiled for its translation into a knowledge-based system? There is an obvious high cost in performing the decompilation, trying to rediscover the reasons back into the experience. Retrospection in a process such as long term learning by experience is difficult. Furthermore, artificial agents would have to re-learn by experience in order to reach again the same level of competence, and thus performance. Although decompilation may be often necessary, and non-compiled knowledge must be expressed, it is important to develop languages in the AI tools allowing to directly map this compiled knowledge.

The difference in behavior between the expert and the novice in real life can be best noticed for instance at the attentional level. The expert rapidly focuses on the most relevant knowledge and then organizes the agenda in a structured fashion. On the other hand, by that time the novice has not infered yet the important discrimination factors; s/he has not dynamically indexed the long-term memory and has not replaced enough search by knowledge.

Thus, we need an appropriate mix of functions and representations allowing both the preservation of the compiled knowledge and the decompilation of other aspects of the expert's corpus of knowledge requiring the representation of lower level, algorithmic tasks. The preservation issue requires the availability of high-level primitives in the system, whilst the decompilation is addressed here by interface issues. In any case, if the expert is aware of available compiled knowledge, knowledge systems should be able to directly capture those heuristics and not force the designer into a low-level representation of the task requiring a decompilation effort. This effort would moreover have to be followed by one aiming at reconstructing the efficient behavior using the system's functions from the decompiled knowledge.

In order to work towards a comprehensive theory for the design of such systems, we chose to approach here the knowledge acquisition issues from the problem-solving perspective. We need to explore how the problem-space of the domain-expert can actually be searched.



Figure1. Recursive problem-solving tasks in the design and use of knowledge-based systems. The knowledge environment designer has to solve the problem of building the system. To do so, information on how knowledge designers solve the problem of building a knowledge base is useful. In turn, the application solves the problem. The knowledge designer is the domain expert in this model.

Since we assume that the domain-expert is performing a problem-solving task, we may consider what techniques are being used (or should be) from the computational point of view. In the very initial phase, s/he may not have a well structured goal. Either the problem-solving functionality is unknown to the expert or no particular task or domain has been identified yet. Using the tool thus consists more in an exploration than in an actual design. However, the tool must be able to show its functionality and potential. Past this early phase, the task of the expert is to implement a particular problem in a given domain. The absence of clear heuristics and in particular experience with the tool from which to draw upon will gear the expert into the use of domain-independent methods such as means-ends analysis (creating subgoals and achieving them). As the user becomes experienced with the tool, s/he can draw upon this experience to design new applications. Therefore, analogical reasoning becomes relevant. Such approaches may be combined with others such as plan instanciations. The two main factors to be considered are: there must be an evolution in the type of problem-solving techniques used and elicited by the tool, in order to gain performance.

The design of the system's specifications must be influenced by this information processing. approach (fig 1). We distinguish two overall types of methods, the weak ones address domain-independent knowledge acquisition while the others are task-dependent. In the present study, we are concerned mainly with the weak, domain-independent approach, and the progression towards domain-dependent methods that the tool must favor.

### DECOMPOSITION OF THE PROBLEM SPACE

The designer's problem-space is made of all the knowledge available to the expert. Thus, it is apparently too large to be considered, both in nature and in size. In order to progress, we should decompose this problem-space in subspaces corresponding to particular functions in the design task. The following decomposition into three subspaces allows us to formalize some important design issues.

#### The application space

The application space corresponds to the corpus of domain specific chunks of knowledge used by the expert. From the specifications point of view, it addresses the representational component of the design task.

**Representation.** From the point of view of the tool's design, unless one voluntarily limits the type of problems to be addressed, the system must support various, though interrelated, knowledge representations. There is of course no need to work at the level of the actual "thing", as if the images compared by the mind were compared at their primary level of perception on the retina. This approach does not make sense from the computational viewpoint. Thus, the lowest level of representation should be of a high enough symbolic level, such as *objects* with a local and global organization, describing the percepts (instances) or concepts (generalizations) as sets of properties and relations. Such static representations must coexist with more dynamic ones from various types of constructions to rules, which by definition give a sense of cognitive progression and problem-solving to the developer. The representation environment can be seen as a multi-dimensional space of structures related one to the others. For instance, rules address objects, perform pattern-matching on their multi-level organization, modify their characterictics and their values.

Whether a unification of the mental and physical models is possible or even desirable is difficult to assess and is case-dependent. The observer-dependent components of the knowledge base can be rules as well as objects. Previous studies (Larkin 1983) have shown the shift in the nature of the representation from novices to experts. What we would like to stress is the need to be able to represent the psychological representation of problems in the expert's mind.

In modelling processes for example, the description of the device can often be more or less abstracted and reduced to the main events which may occur and their impact on the system. This description can be very dependent upon the expertise and experience of the domain-expert. More classic simulations aim at linking an intelligent system to or using the language to program a very precise description of the system's functioning. The result is a greatly increased difficulty in both the debugging and the maintenance of the system. The implementation of the expert's model of the key events and functions can be as powerful and provide a more dynamic, maintainable model.

Another example of adaptiveness of the representation is the composition of models in terms of instances and classes. These notions are useful to direct the transfer process from the expert to the system. However, one must be able to reflect, in the organization of those structures not only material, physical relations but also psychological manipulations. Down-inheritance allows one to describe the hierarchical physical nature of things. Whilst the description may be correct, it use by the mind looks often more like a heterarchical world. When one sees the very front of a car one can surely infer the color of that car before seeing the entire object. The system allows rapid transformations from prototypical objects to generalizations and vice-versa, as experts do. The dynamics of the structural representation is a graph of uni- or bi-directional relations. This flexibility is essential in reproducing the use of concepts and examples in reasoning (as well as metaphors), and in establishing correponding learning mechanisms.

**Structure editing and compilation.** Because of the variety of domains, tasks and even approaches to similar tasks, we need to insure a basic flexibility in the editing mechanisms, so that the designer can talk about objects, classes or properties in a rule before creating the latter if s/he has a better sense of the dynamics than of the overall representation. Indeed, the high level implementation - not exclusive of the rest, is best allowed by not forcing any particular framework of development such as static knowledge first followed by dynamic operations on the knowledge space. Rather, an approach allowing a permanent shift between the different aspects of the domain (mainly the static and dynamic components) should best capture the natural equilibrium betweeen those concepts in the actual task.

Hence, being able to extract as much information about the structures from the rules at the compilation is critical, for the rule compiler automatically assumes part of the work at the descriptive level as well. Furthermore, the rules applied to previously created objects should be able to bring about important information about the more precise nature of objects and their properties. We call this concept of editing objects and classes from their description in

rules *cross-compilation* of knowledge structures. The objective is to adapt the system to the great fluidity and complexity of the overall model of the task, reflecting both the mental and physical components of the model. Furthermore, in order to allow a continuous development, incremental compilation of structures by the tool is necessary.

**Selective interface.** Whatever the application domain, it is necessary to capture the chunks of knowledge involved in the task to be modelled. These chunks can be rules, schema, frames or other representation. In any case, there is, for the domain-expert, the task of finding or discovering what those chunks are and entering them into the system. Thus, the primary work takes place at what we call the *microscopic level*. Even though the designer certainly has in real life a global vision of his/her task, this perception may be lost when it becomes necessary to work at the microscopic level.

It is thus important that the tool allows one to take a global view at the space of knowledge elements, objects or rules, that is to say automatically reconstructs the structural links between them, thereby generating a global image of the corpus of knowledge (as a whole or of part it). To provide such functions, one must take advantage of the selectivity of visual perception (Arnheim 1969) and its powerful global/local relation. It maintains the correct focusing of the attention on more or less complex arrangements of chunks of knowledge. Such visual thinking provides a powerful incentive to create new chunks and structure the knowledge.

Provided that the visual tool is part of an interactive environment, the visualization greatly enhances the tasks of unification, organization and integration of knowledge. It also allows the reformulation of parts of or entire problems. This elicitation factor is even more important when one knows that mental representations of a problem are by essence more fluid, changing more often than the physical ones.

NEXPERT allows one to represent the macroscopic structure of the knowledge chunks via the *Network* functionality. In general, the meshed networked structure of knowledge does not only result from the fact that certain concepts share attributes in their definition, but also from the way they are processed by the rules. Thus, there are *functional* links between the elements of a knowledge base, as follows in the figures showing examples of the Network of rules (fig. 2, 3).



Figure 2. Deductive navigation in a knowledge-base (rule plane) using the NEXPERT network.

The possibility of *navigating* through a knowledge base by simulating/viewing inference functions (deductive/backward and evocative/forward), and the structural organization of objects is crucial in the understanding of the knowledge processing task by domain-experts. The evocative navigation simulates some of the forward processing notions, such as the immediate agenda set-up upon volunteering of one or several data values. Furthermore, in both ways, the declarative conflict resolution method in NEXPERT can be also visualized, as well as the object representation.



Figure 3. Evocative navigation. Clicking on the CRT\_and\_KDU item with the forward icon displays the goals or subgoals in which this attribute is involved.



Figure 4. Overview of a navigation in parts of a knowledge base. It was noticed that domain-experts recognize the visual patterns of rules belonging to their knowledge base, facilitating the use of this zooming mechanism.

The cognitive continuum between the user and the knowledge thus results from two important elements: the representation must attract the user's attention with the highest gain, and the actions resulting from this observation must be rapidly integrated.

At the same time as the interface can correctly keep the designer's attention on the incremental editing of knowledge, lower-level interface optimizations must prevent the perturbation of this effort by distractions due to a bad display organization and/or non-optimized click-distances.

#### The Methodological space

The methodological space addresses the system's ability to model the behavior of the expert, or the dynamics mental model in terms of available inference functionalities. It thus addresses the functional component of the overall problem-space.

Human reasoning vs. theorem-proving. There are fundamental issues to tackle as well concerning the logical mechanisms available to the developer. The philosophical attempts to establish a relation between formal logic and identity has now been overcome by a large body of work, and new approaches to logic tend to formalize a more psychological approach to logic (McDermott and Doyle 1980, Doyle 1982, Harman 1985). Flexibility in the representation of facts and logical dependencies is thus necessary. In theorem-proving situations, the facts cannot be contradicted because of their universal validity, whilst in human reasoning, new data come and contradict previous inferences all the time. Inferences at the mental level reflect the application equivalence relations based on the permanent matching to the environment. On the contrary, formal logic approaches do not clearly distinguish facts from theorems and are search-driven instead of being event/concept-driven. We feel that the logic must be somewhat more adaped to the non-monotonic nature of human reasoning. More formal approaches can be applied to local subproblem spaces to which they are relevant.

The main charactistic we emphasize is the need for the knowledge designer to be able concentrate on the implementation (mapping) of his/her thought process rather than on having to focus permanently on the consistency of the underlying representation language, as in prolog-like systems. The result is an acyclic graph of rules.

**Agenda levels.** As said above, a major way of distinguishing an expert's behavior from that of a novice is to study the focus of attention, which reflects a compiled set of heuristics to perform an optimized search in the problem-space.

In NEXPERT for instance, the *context* allow sets of related rules to be linked together by what are called *weak* links. Such links represent shifts in the focus of attention, products of highly compiled knowledge. They are not symetric and thereby allow the easy sequencing of a job, the representation of intuitive, inductive knowledge (which does not need to be decompiled). The first result from a computational and knowledge design point of view is an important economy in the number of rules used to represent the problem.

Different levels of granularity in the focus of attention can be distinguished, from the prompting order directed by knowledge of a cost efficiency measure to the selection of overall goals on experience-defined criteria. These agendas are easily implemented using declarative mechanisms at different levels of the representation. This corresponds to a controllable, dynamically updated, declarative agenda, opposed to the lack of accessible control structures of lower level languages which operate while generating of large quantities of eventually useless information. Sets of rules can be as small as one desires.

In order to insure all levels of implementation, the system must integrate various

methodologies for such mappings as well as more classic issues, from agendas to prodution-system-like mechanisms. Figure 5 shows an architecture where different *granularity levels* for the focus of attention are used in the same knowledge base in the NEXPERT system. It shows a sample architecture of a knowledge base, with its different levels of knowledge processing.

Each ellipse encircles a *knowledge island*, an independent set of rules, sharing no attributes with one another. Within each island, progression occurs in an integrated forward (fwd) and backward (bwd) fashion. The islands are linked by a forward indexing mechanism, the *context*, indicating future focus of attention. New directions of search can also be selectively induced as a result of right-hand side (RHS) actions. The flow of control from one island to another or the effects of any RHS action can be controlled from the rules themselves. For instance, the consequences of RHS actions may or may not be investigated. The overall result is the possibility to organize the search according to psychological evidence. The fluctuations of the focus of attention are not necessarily trivial and declarative access to this mechanism should enable the implementation of many variations.



Figure 5. Management of the focus of attention by declarative heuristics. The top-level flow of control from the original task is concerved by the system. Horizontal (e.g. scheduling) as well as hierarchical (refinement) or mixed representations can thus be implemented.

The concepts of contexts and knowledge islands represent the type of shortcuts necessary to embody intuitive reasoning without having to go back to a description of the physiology of the problem or of the causal mechanisms (deep knowledge). Contexts act as declarative heuristics limiting the search space. However, within each island, the problem-solving task can be quite similar to more classic systems. Still, inside each island the focus of attention can be controlled as well, simply at a lower granularity scale.

Thus, the control of the flow of information must be accessible by the use of "cognitive primitives": local and global priorities, context links, revisions, conflict detection... At the same time, more classic approaches, closer to the algorithmic calculations and searches, such as Prolog or OPS can be used to perform subroutine work in adequate, relevant problem subspaces. Thus, cognitive primitives represent the high-level programming, embodying control mechanisms, while languages that lack control structure should be used to perform subtasks (recognition, filtering, tree searching etc...) of a more algorithmic nature, as treated by production systems.

The mosaic conception of uncertainty treatment. When solving complex problems or assessing complex situations, different qualities of knowledge are often applied. Regarding uncertainties in the knowledge itself or from the user, this knowledge mosaic must be treated with a corresponding mosaic of methodologies, for no uncertainty treatment method found to date has a universal value. Thus, it is necessary to be able to implement different methodologies within the same knowledge base. From the designer's standpoint, limitations in the uncertainty treatment prevent the implementation of adequate mechanisms. Rather than orienting the search problem of the designer, it perturbs it.

For instance consider a situation assessment requiring the use of empirical test results. These can be processed using conditional probabilities. However, the same treatment may be unsuitable for other elements of the knowledge base which do not lend themselves to this type of analysis. More arbitrary techniques such as certainty values or knowledge based techniques such as default assumptions provided by the domain expert(s) to solve particular problems may become relevant for other parts of the same knowledge base. In the latter case, uncertainty is solved by situational rules derived from an expert's experience. Furthermore, as mentioned previously, logical dependencies are necessary in order to backtrack along previous reasoning paths, to revise beliefs and conclusions. Uncertainty design results from the availability of different tools to adapt the methodologies to the nature of the knowledge being treated.

### The processing space

The processing space is the problem-space in the design task concerned with the execution of the program. It precisely addresses the relation between form and function, and makes a bridge between the two subspaces above.

The runtime environment for the designer must allow a selective reduction of the overall search space, setting its boundaries by allowing to interactively discover inconsistencies, irrelevances, redundancies as well as potential unifications and refinements.



Figure 6. This display of a Rule Network graph shows the state of part of the body of knowledge during a session. The icons and different fonts and styles indicate the various states of the conditions and goals. Rules can be edited during the session by calling the rule editor which then compiles modifications incrementally.

Insuring the cognitive continuum between the user and the system requires that the relevant design tools remain available during the verification phase, in which the user tests the current state of the knowledge base. The notion of example and trial execution is central to the development process, addressing the refinement and verification functions. Execution of the inference process based on the available knowledge at any time focuses the designer's attention on selecting the most relevant chunks of knowledge.

While the use of the Network in a non-interactive manner favors a rapid growth of the corpus of knowledge, the same visual information obtained during execution (fig 5) is of a different order: it sets boundaries to the designer's search space, and prevents the designer's inference process from drifting to unnecessary subspaces of knowledge.

Hence, it is necessary to integrate these functions into a coherent psychological environment. The execution phase brings new information which needs to be rapidly included in the system. Since such modifications stemming from testing the dynamics of the system's knowledge are of a particular order, they should be tested as quickly as possible within the same execution environment. This functionality requires an incremental compiling mechanism (creating, modifying or deleting a rule) available during the execution. Coupled with dynamic visualization, this allows one to maintain the designer's focus all along the debugging of knowledge. These facilities constitute an interactive knowledge design environment.

Yet another important issue addressed in the processing space is the tracking of inconsistencies. When knowledge based systems use formal logical statements, thereby confusing rules with theorems, inconsistencies exist only in the internal anatomy of the knowledge (theorem) base. The consistency checking is therefore mostly independent of the problem solved, and pre-processed according to laws and constraints which may have little to do with the nature and human treatment of the problem. Because of the necessity to represent reasonings such as default reasoning or to live with inconsistencies which are actually dynamically solved by the data of the problem, the high transparency of the execution phase is a critical asset for the design environments. Inconsistencies are incrementally discovered.

#### METHODOLOGICAL PROGRESSION

#### Weak methods, analogy and learning

Considering such powerful techniques, we can now come back to the actual search process taking place in the expert's mind. Initially, the expert is lacking knowledge on how to make a knowledge-base and must rely on an intensive exploration of the tool's capabilities. This will involve the use of the many different mechanisms but can only be useful if certain properties such as incremental compilation and fully interactive environments are available, with the smallest amont of low-level tasks (variable declaration for instance which can be handled automatically as in NEXPERT from reading the structures). In this first phase, the approach is weak, in the sense of weak methods for problem-solving, i.e. domain-independent (Laird and Newell 1983). At this level, the interface issues as well as the format and editing mechanisms of the knowledge structures are critical to gain acceptance and keep the attention focused.

The tool is then supposed to elicitate not only the extraction of knowledge from the domain expert, with the incremental discovery of new knowledge chunks and refinements, but also to trigger shifts in the approaches to the construction of the knowledge-base, which will reflect a better understanding of the tool and its use to formalize a problem. We shall not dwell here upon the obvious practice effect on the use of the tool's text and graphic, menu-oriented interface. One important thing in this regard is of course the adaptiveness of the commands' access as practice is acquired and most skills are built. We are concerned here rather with the internal representation and approaches of the knowledge design task in the designer's mind. The visual navigational tools have the major advantage of combining a macroscopic reconstruction and a complete description of the world of inferences and structures involved in the task. The visualization helps to develop models which are symbolic descriptions of rules or objects which perform certain functions in given contexts. This occurs with any individual learning a tool's use, but we are concerned here with the rapid facilitation of this process in order to make the reasoning functionality available to domain experts directly.

Thus, there is a clear progression from what is initially a weak, domain-independent approach to more structured, domain-dependent thinking involving in particular the notions of chunking and reasoning by analogy. Chunking (as in Rosenbloom and Newell 1985) is facilitated by the interactive interface which favors the unification of elements of knowledge as part of the refinement process. While chunking may result in the generation of new structures, the interface also favors the exploding of too compact chunks, the reverse process. The interactivity of the tool allows one to take advantage of those mechanisms.

At a somewhat higher level, reasoning by analogy is also enhanced by the tool. It applies to the use of control structure mechanisms such as the various levels of agenda in the architecture, as well as analogies at much lower levels of granularity such as "copying and editing" a rule acting as a template.

The following text is an example of an account of a knowledge acquisition process by a user. The domain expert in chemical engineering had *no* prior knowledge of AI or programming.

### Expert System Building steps using NEXPERT™

1. General Description of the problem

This expert system (ES) is a troubleshooting-diagnostic program for quality defects on a production line. In the past, the operators were instructed by a line specialist. This system attempts to replace the line specialist for most of the routine problems, while recognizing the possibility of a difficult problem that might still require the line specialist.

The operator will identify a defect and then use the ES to correct the production line. He will be guided through several steps to correct the problem, anyone of these steps or actions might correct the problem, but there is no real certainty which of the actions will do it. Therefore it is important to constantly check to see if the action was carried out and if the problem was corrected.

### 2. Identify Knowledge Islands

At first I implemented a set of rules from an old troubleshooting guide. I then realized that this problem was fairly nicely divided into five knowledge islands. I spent some time learning how to organize the overall problem. It took me awhile to grasp how the rules could both be forward and backward chaining and then to understand the use of context to control 'weak links'.

### 3. Develop Meta rule for ordering the rules

Concentrating on the first knowledge island, I worked out a system for setting the order that the rules would be fired in. At first I used categories, but I then decided that it would be better to use a meta-rule to control the rule order. Once I had debugged this structure I cloned the same structure for the other knowledge islands.

### 4. Develop method for cycling each rule

After demonstrating the initial ES several people commented that in the real world the expert will verify if the action was in fact taken and if it worked, until a positive response is given he will not procede. Therefore, I developed a method for cycling each question and checking to see if the problem was solved. I again debugged this procedure on the first knowledge island and then cloned the procedure for the other islands.

Table1. Protocol obtained from a designer with NEXPERT<sup>™</sup>.

A rapid analysis of the report in Table 1, paragraph by paragraph, allows us to distinguish some interesting mechanisms:

(i) description of the nature of the task at the basis of the application.

(ii) exploration phase based on previous models and design of an overall architecture, from

the focus of attention viewpoint.

(*iii*) work on a sub-problem using specific methodologies provided by the tool (reset, categories, meta-rules. Then notion of "cloning", or mapping the same structure to another part of the problem. This is the simplest level of analogical reasoning, where the problems (or subproblems) are very similar.

(*w*) modifications are suggested at the cognitive level (cognitive debugging) concerning the temporal course of the task. The overall progression described before is re-used (analogy) and the system is updated. The notion of "cloning" is adopted as a problem-solving technique. The problem-solving strategy used previously is mapped to the current problem, in a derivational analogy-like process. A technique has been acquired for solving a particular problem and is re-instanciated as a whole after having been identified as relevant.

In reporting the structure of the knowledge base, the designer also describes the use of a technique, or operator recursively applied recursively. In this instance the system can use the same control strategy at different scale levels of the architecture. The implementation is made possible by means of the reset operator and control over the propagation of new information.

Finally, it is of interest to note the learning process exposed by the expert. We can observe the shift in methodology and representation from what seems to be a type of exploratory approach to an experienced based approaches, as in cognitive development (Piaget 1954, Carey 1984). In this particular case, the overall architecture built here is currently being transposed to a similar task to be modelled.

The protocol described in Table 2 by a knowledge designer working on a complex task reflects both similarities with the other one and another aspect of the knowledge design task. The initiation of the project with *textbook knolwedge*, rapidly followed by an implementation of the expert's knowledge. The textbook knowledge serves as a primer example on which the initial learning takes place. The interactivity of the tool is critical in this phase, for the domain-expert to focus on the tool and obtain a sense of the problem-solving methodology.

1. Map sequential textbook steps into intermediate hypotheses (DE)

2. Determine use of certainty factors (how, what for, thresholding, techniques (DS, Fuzzy sets...)) (DE)

3. Code and debug first set of rules (DE)

4. Demonstrate to human experts (DE)

5. Select the right approach for easy extension (DE + AI eng)

6. Determine operator-prompting sequence

7. Generate Show and Apropos files (DE, long task)

8. Code and debug a second set with a new architecture taking into account more assumptions, concepts and ideas (DE + AI eng)

9. Categories to achieve prompting sequence (DE + Al eng)

10. Determine appropriate strategies (DE + AI eng)

11. Extend second set of rules (DE)

12. Debug with intensive use of the Network, this task was extremely simple (DE + AI eng)

13. Multiple level Apropos, multiplication of the second set into many such sets to deal with pther aspects of the problem and complete the knowledge base, and linking the sets (DE).

14. Demonstrate to human expert (DE)

The expert did all the coding. The AI engineer identified problems in the way the DE implemented the problem and was there to provide advise and debuggin help. Furthermore, the AI engineer did not have to know much about the application.

 Table 2. Protocol of a development by a domain-expert, illustrating the major role domain expert in the design process. (DE : domain expert, AI eng: AI engineer).

This example also illustrates the replication approach, allowed by the ability to test and understand the behavior of the system on small but representative sets of rules. Moreover, and this is the second aspect, the tool is in the hands of the domain-expert and the AI knowedgeable person intervenes by "looking over the shoulder", providing (important) advise. This transfer of power to the expert is critical in the end-result mapping of its thought process.

**Elicitating problem-solving methodologies.** We are thus concerned here with the optimization of the elicitation of such process by means of automated tools. The examples above concerns specific tasks to be modelled. Other types of applications have been built where the same type of reasoning process takes place with the expert. The ability to provide quickly and concisely a report on the making of an application by an AI novice is illustrative, in our sense, of the "mapping" effect looked for. The extensive use of the Network also recalls the notion of "perception of analogies" put forward by Polya (Polya 1945) in problem-solving activities.

Such behaviors can only be elicitated and maintained if the tool's mechanisms are close enough to the cognitive mapping concept and is thus based on very high level primitives, the rest of the knowledge invoved being dealt with with classical methods (by the same system though).

It is thus indispensible to provide not only powerful interfaces but an underlying language allowing these high level implementation. If most of the work is concerned with making sure that the language's rules and low level indiosyncrasies are consistently used, it is that much less work done at the high-level. In other terms, if the reconstruction of the control structure requires programming in a forbidding language, the tool will remain as obscure as it can be. Hence, the need for high-level primitives such as the context and other declarative accesses to the inference mechanisms, which can be quickly understood and manipulated by the domain expert (because they are close to the cognitive description the expert has of the task).

Thus, we observe a progression in the problem-methodologies used by the domain-expert. Much research and implementation remains to be done though in this direction, to find more adapted techniques. The shift in methodology is typical of a basic learning process, involving first weak-methods and progressively showing capabilities in reasoning by analogy or discovering macro-operators for the design of a task. As those techniques are applied, the body of knowledge on how to construct a knowledge base increases and can be used for new, different domains, allowing the rapid identification of similarities between tasks and domains. The aim is to considerably reduce the overall learning curve.

#### Automated integrated tools

Such techniques could be associated to automated interviewing tools for knowledge acquisition. The work based on Personal Construct Psychology for interactive interviewing (Boose 1985, Gaines 1981) illustrates the progress of such techniques. A structural analysis can yield rules used by an expert system. However, there cannot be any *a priori* as to how the rules should be processed. A forward-only or backward-only inference mechanism would be too "prejudiced" to allow a correct cognitive debugging. On the other hand, a more opportunistic system with symetric rules as in NEXPERT will allow one to fully test the meaning of the resulting rules and , along with the visualization mechanism, will allow the progressive refinement (unification, selection etc...) of the initial knowledge (Shaw and Gaines 1986). Also relevant are automated techniques based on the analysis of reports from skilled individuals (Nisbett and Wilson 1977, Ericsson and Simon 1983).

Furthermore, learning techniques performing comparable tasks (Winston 1980, Rosenbloom and Newell 1985, Carbonell 1985, Cheng and Carbonell 1986) could be

progressively included into the tool so as for the latter to automatically analyse, from its perspective, the structure of the knowledge and adapt its reasoning methodologies. Learning mechanisms should not result into the implementation of a black box but should also be interactive. Task-dependent knowledge can also be used to guide the exploration of the problem, as in classification applications (Eshelman and McDermott 1986).

#### CONCLUSION

While graphical interfaces were not studied in previous work (Card et al. 1983), they actually play a fundamental role in the design of intelligent systems. As in the case of automatic algorithm design (Kant 1985), the problem-solving approach is a powerful heuristic for building both perceptual and automated tools. New issues in the psychology of the human-computer interface arise from systems as the one described here, leaving plenty of perspectives for a hardening of the psychological approach (Newell and Card 1985).



### HIGH-LEVEL PROGRAMMING

Figure 7. The distance between the problem-solving activity and the model being implemented must be minimized. While low-level programming requires decomposition of the problems into many, small elements of knowledge and then the reconstruction of the control structure, high-level programming attempts to preserve via elicitation and high-level operators a maximum of compiled knowledge.

Based on a cognitive analysis of the task of formalizing knowledge, key issues in the design of knowledge-based systems can be identified which limit the need for the knowledge engineer and gears the technology towards the status of a functionality per se, directly available to the domain-experts. The resulting concept is that of high-level programming (fig 7).

In general, progress is sought both on the impact of artificial systems on the reasoning of the human experts and on computational methodologies allowing an implementation of the expert's design of a task as opposed to an implementation of the expert itself.

#### ACKNOWLEDGEMENTS

The author thanks Albert Gouyet, Bruno Jouhier, Patrick Perez and Chris Shipley for most helpful comments on earlier drafts. Special thanks also to Jaime Carbonell and Brian Gaines for some insightful discussions. This research was made possible by the support of Neuron Data.

### REFERENCES

Boose J.H. (1985) A PCP-based method for building expert systems, Boeing Computer Services Report BCS-G2010-24.

Carbonell J.C. (1985) Derivational Analogy: a theory of reconstructive problem solving and expertise acquisition, *in* Machine Learning II :371-392, Morgan Kaufmann, Los Altos CA.

Card S.K., Moran T.P. and Newell A. (1983) The Psychology of Human Computer Interaction, Lawrence Erlbaum Associates, Hillsdale, NJ.

Carey S. (1986) Conceptual Changes in Childhood, MIT Press, Cambridge MA.

Cheng P. and Carbonell J.G. (1986) The FERMI system: Inducing Iterative Macro-operators from Experience, Proceedings of the Fifth National Conference on Artificial Intelligence, Morgan Kaufmann Loas Altos, CA 1:490-495.

Doyle J. (1983) Some theories of Reasoned Assumptions, Technical Report, Carnegie-Mellon University, Dept. of Computer Science, CMU-CS-83-125, Pittsburgh PA.

Ericsson K.A. and Simon H.A. (1984) Protocol Analysis, MIT press, Cambridge, MA.

Eshelman L and McDermott J. (1986) MOLE: a knowledge acquisition tool that uses its head, Proceedings of the Fifth National Conference on Artificial Intelligence, Morgan Kaufmann Los Altos, CA 1:950-955.

Gaines B.R. and Shaw M.L.G. (1981) New directions in the analysis and interactive elicitation of personal construct systems, *in* Recent Advances in Personal Construct Technology, Shaw M. (ed) Academic Press, New York NY.

Harman G. (1985) Change in view, MIT Press, Cambridge MA.

Kant E. (1985) Understanding and automating algorithm design, Proceedings of the Ninth International Joint Conference on Artificial Intelligence, 2: 1243-53, Morgan Kaufman Publishers, Los Altos CA.

Larkin J. (1983) The role of Problem Representation in Physics, *in* Mental Models, Gentner D and Stevens AL (Eds), Lawrence Erlbaum Associates, Hillsdale NJ.

Laird J. and Newell A. (1983) A Universal Weak Method, Technical Report, Carnegie-Mellon University, Dept. Computer Science, Pittsburgh PA.

McDermott D and Doyle J (1980) Non-monotonic logic I, Artificial Intelligence, 13:41-72.

Newell A. and Card S.K. (1985) The prospects for psychological science in human-computer interaction, Human Computer Interaction, 1: 209-242.

Nisbett R.E. and Wilson T.D. (1977) Telling more than we can know: verbal reports on mental processes, Psychological Review 84:231-259.

Piaget J. (1954) The construction of reality in the child, Basic Books, New York NY.

Polya G. (1945) How to solve it, Princeton University Press, Princeton NJ.

Rosenbloom P.S. and Newell A. (1985) The chunking of goal hierarchies, in Machine Learning II :247-288, Morgan Kaufmann, Los Altos CA.

Shaw M.L.G. and Gaines B.R. (1986) Interactive elicitation of knowledge from experts, Future Computing Systems, 1: *in press*.

Schank R.C. (1982) Dynamic Memory, Cambridge University Press, London.

Winston P.H. (1980) Learning and Reasoning by Analogy: the Details, Technical Report, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, AIM 520.

NEXPERT<sup>™</sup> is a trademark of Neuron Data Inc. Figures 2,3,4,6 obtained using NEXPERT<sup>™</sup>, with the permission of Neuron Data.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

DRAFT

# Ontology and Inventory: a Foundation For a Knowledge Acquisition Methodology

Stephen Regoczei Computer Studies Trent University Peterborough, Ontario K9J 7B8

Edwin P.O. Plantinga Department of Computer Science University of Toronto and Redeemer College (Hamilton, Ontario)

> Ontology recapitulates philology. James Grier Miller as quoted by W.V.O. Quine

If you can't build a model of it, it isn't true. Buckminster Fuller

The science of personal relations is not assisted by the fact that only a few psychologists are concerned to discover valid personal ways in which persons, and relations between persons, can be studied by persons. R.D. Laing

### **1** Introduction

Systems analysis, software requirements definition, conceptual database design and knowledge engineering for expert systems can all be viewed as activities which have essentially the same structure. The goal of all these processes is to acquire the world knowledge possessed by an informant who, depending on the situation, may be called an expert or a user. Through the mediation of at least one other person, generally called the analyst, programmer, or knowledge engineer, the process culminates in a "smartened-up machine".

The bottleneck in developing knowledge-based software is the knowledge acquisition (KA) phase. Given the importance of this problem, why is the literature in this area so small, especially when compared with the literature addressing the knowledge representation (KR) issues?<sup>1</sup> We suggest three reasons for this phenomenon.

The first reason is obvious. The problems associated with the representation of knowledge are easier to deal with than the problems of "acquiring" knowledge. Representational issues are also easier to write about and discuss since they ignore the informant-analyst relation. Here is the first reason why the literature on knowledge representation is so much more extensive than the knowledge acquisition literature.

The second reason is slightly more complex. KA is a real dollars and cents issue in industry, that is in places where software must be pushed through the door within budget and on time. In most industrial settings, the analyst needs those crucial bits of knowledge which the informant possesses. In order to construct a working and workable knowledge-based system, this knowledge must be acquired.

Contrast this with a typical situation in the academic world. An academic Computer Scientist, engaged in research, is not producing software for some user community. The academic knows what he wants so there is no need for a KA phase. In the academic world, the analyst and informant are one and the same. The key issue from this point of view is representation. The academic wonders how he should encode or represent the knowledge in his head in such a way that it is useable by a machine.

When academics publish in the domain of knowledge-based systems, they, in clear conscience, address those issues which they consider to be crucial. Although we agree with academics that there are important representational issues, we also side with those in industry who stress that the acquisition of knowledge is the crucial problem at present. However, people in industry do not publish their results for fear of losing their competitive advantage in the marketplace. This, we submit, is the second reason for the imbalance in the favour of representation in the knowledge engineering literature.

But there is a third reason. The view of the human mind, which is dominant in our culture, is simply too static to explain the difficulties analysts encounter in the KA process. Systems analysts frequently complain that "the user keeps changing his mind." We believe that this is to be expected, that the user "changing his mind" is the rule rather than the exception. Daniel McCracken and Michael Jackson have argued that

<sup>&</sup>lt;sup>1</sup>A good recent summary of KR is Ronald Brachman's and Hector Levesque's Readings in Knowledge Representation.

Systems requirements cannot ever be stated fully in advance, not even in principle, because the user doesn't know them in advance—not even in principle. To assert otherwise is to ignore the fact that the development process itself changes the user's perceptions of what is possible, increases his or her insights into the applications environment, and indeed often changes that environment itself. We suggest an analogy with the Heisenberg Uncertainty Principle: any system development activity inevitably changes the environment out of which the need for the system arose<sup>2</sup>.

We take these comments by McCracken and Jackson very seriously. How can someone construct a system for a user if the user keeps changing his or her mind? We take this comment as a challenge and in response to this challenge, we have developed our KA methodology.

The human mind is not well understood. When confronted with a poorly understood phenomenon, people invariably search for metaphors or analogies whereby they can think of the abstract entity in terms of a more concrete entity. We think that it is a distinct possibility that many of those currently involved in knowledge engineering view the human mind as an expert system. Using this metaphor, the KA problem becomes one of acquiring the production ruleset of the informant. Presumably this ruleset is fairly static. An analyst who works with the *expert system metaphor* becomes upset when the informant "changes his mind." We believe that it is time for a new metaphor: mind as an anthill in which the ants are constantly in motion. We develop this metaphor in section 3 of the paper under the heading, "The Mental Models Hypothesis."

To give a comprehensive account of KA, we have to depart from the standard descriptions of KA practices as published in the academic Computer Science literature. We believe that it is essential that the "architecture" of the informant and the acquirer be considered. While studying the "architecture" of these persons could be considered to be outside the bounds of Computer Science, it is very much in the domain of Cognitive Science and therefore not at all out of place in a paper dealing with issues of Artificial Intelligence and expert systems.

# 2 Overview of the Methodology

There is a group of historians, philosophers, and literary critics in our culture who call themselves post-structuralists<sup>3</sup>. Post-structuralists ask questions such as the following:

- 1. When history is written, whose history gets written? Just the kings' and queens', or are the historians also prepared to write histories of the poor, the insane, and the prisoner?
- 2. Who decides what metaphors mean?[11]
- 3. Who gets to speak? Is the madman allowed to speak? Is the pauper's voice heard?

<sup>&</sup>lt;sup>3</sup>Daniel D. McCracken and Michael A. Jackson "Life Cycle Concept Considered Harmful" Software Engineering Notes, Vol 7, No 2, April 1982, p. 31

<sup>&</sup>lt;sup>3</sup>The most celebrated member of the post-structuralist camp, Michel Foucault, died last year.

The post-structuralists object that those who hold the power impose their discourse on those who are powerless.

We believe that this discussion is relevant to KA and software development. Software developers (those who hold the power) can impose their discourse (in the form of a user interface) on the powerless (the user)<sup>4</sup>. We believe the informant or user is to be respected; we believe the words of the informant are to be taken seriously. Who is allowed to speak? The informant, but only for so long. System developers have a job to do and that job must be completed in a finite period of time and within budget. Our methodology seeks to be realistic by providing the best possible system description within the constraints imposed on the project and all the while taking the words of the informant very seriously.

Our methodology relies on interviewing, diagramming, and conceptual analysis. An analyst interviews an informant and then diagrams the sentences produced in the interview in a graphic notation. When the analyst and informant have agreed that the graphical notation for the domain of discourse is complete, the analyst performs conceptual analysis [18,17] on the graphs. This leads to an ontology, a list of entity types, and an inventory, a list of instances of these entities.

The purpose of the ontology is to reduce the inherent fluidity of the KA process which McCracken and Jackson have described [8]. The ontology list, that is, the list of entities that "exist", defines the entity population of the domain of discourse. The document, in a sense, is a contract between the informant and the analyst that these are the only entities that will be talked about in the future and that these are the only entities the machine is expected to be able to recognize. Without such a contract, the expectations placed upon the machine are infinite. This phenomenon is well known to constructors of expert systems. After having constructed a well-functioning system in a limited problem domain, users of the system are shocked that the system cannot, for example, fix flat tires.

The ontology gives to both constructors and evaluators of expert systems a reasonable framework that defines what "exists". An ontology is, in fact, a definition of what constitutes the domain of discourse.

Trying to construct an expert system on the basis of collecting rules or propositions is only possible in clearly defined knowledge domains such as medicine. If the knowledge domain is not clearly defined and codified, so that it is possible to state clearly what is in and what is not in the domain, then project management, budgeting, time schedules, and evaluation criteria become uncontrollable. Not only do users change their minds, so do managers. As a matter of sound practical project management practice, the ontology defines the problem domain in a field not previously codified.

<sup>&</sup>lt;sup>4</sup> "Powerless" users are adaptive organisms and are fairly reluctant and slow to rebel against poor interfaces. The issue of developing a user interface with minimal imposition on the user is discussed in "Towards Automated Methods for Developing User Interfaces" written by one of the authors of this paper, Edwin Plantinga. This is available as *Technical Report* 109 from the Department of Computer Science, University of Western Ontario.



Figure 1: A Model of the Person-to-Person KA Process

### 3 The Mental Models Hypothesis

In figure 1, we show the essential components of the person-to-person KA process based on the mental models hypothesis. The mental models hypothesis states that an individual understands the world by forming a mental model, that a cogniting<sup>5</sup> agent understands the world by forming a model of the world in his or her head.

Suppose you want to form a mental model of Alberta. You cannot look at Alberta because your eyes are not big enough to take in all of Alberta, so what you do is consult a map of Alberta. A map of Alberta is a physical model of Alberta. You fold the map up and put it away. Whatever fragmentary information that remains in your head is your mental model of Alberta based on looking at the map. It is not your mental model of Alberta based on "reality"; you have not seen Alberta—you were looking at a map. But if you do want to look at Alberta, you can get in a car and go for a ride. After the trip, your head will contain fragments of information about Alberta based on the trip; after the trip you will have a

<sup>&</sup>lt;sup>5</sup>We will make use of the word "cogniting" which is a gerund formed from the verb "to cognit". We believe (tongue in cheek) that this is what the French philosopher René Descartes had in mind when he claimed *Cognito, ergo sum.* Through a typographical error, this has become *Cogito, ergo sum.* Agents who cognit are studied by Cognitive Science.

mental model of Alberta based on "reality."

How do different people's mental models get harmonized? Harmonization cannot be done directly—we cannot rub mental models together. What actually transpires is that the informant and the analyst are constantly revising their mental models (MMs). This is the complaint of the systems analyst: the ants keep moving. This revision of MMs takes place under the influence of input such as text. The stochastic process which connects text input with mental model revision is just as little understood as the "reminding" [16] process discussed in Section 8 of this paper.

MMs are rarely harmonized. But in order to construct knowledge-based software, the analyst and informant must come to an "understanding". In the interests of attaining "understanding", the knowledge must become accessible. Although the knowledge in people's heads is not publicly examinable, the KA process requires that the knowledge be represented in a fixed, publicly examinable form. We believe that the publicly examinable representation should be pictorial, or graphical, or diagrammatic. Words are a refuge which either analyst or informant might hide behind to avoid making explicit the contents of the mental models. Our methodology requires that the analyst diagram his "interpretation" of what the informant said. If the informant does not "agree with the interpretation", the process is repeated until both analyst and informant converge on a set of diagrams. We expect the informant to keep "changing his mind"; we expect the ants to keep moving. We believe, however, that confronting the informant with the "meaning" of the informant's words (as interpreted by the analyst) will help him to "make up his mind." Is it not striking that before the recent invention of tape recorders, there did not even exist a fixed publicly examinable way of recording what actually was said?

There are a number of candidate representation languages. Our choice of John Sowa's conceptual graphs [2,3,18,19] as our fixed, publicly examinable form of knowledge representation was made on the basis of the following considerations:

- 1. Conceptual graphs stay close to the structure of natural language used by both informant and analyst.
- 2. Conceptual graphs are a clear notation in which to build models of MMs for public examination.

Other obvious choices are KL-ONE graphs [1] and the object-role information model of ENALIM (Evolving Natural Language Information Model) diagrams used in Control Data's Information Analysis methodology [9].

# 4 Providing Machines with Mental Models

One of the advantages of Sowa's conceptual graph notation is that the graphs (in their linear form) are directly machine representable. If the analyst can diagram the text which he receives from the informant, and if the informant can agree with the diagrammed representation, and if the diagram can be programmed into the machine, then we can provide the machine with

5



Figure 2: A Model of the Knowledge Acquisition Process

a mental model. This process is shown in figure 2 and we believe that this figure captures the essence of what any KA methodology must provide.

Sowa's notation is preferable over other notations for the following reasons:

- 1. The translation of a conceptual graph in its pictorial form to its linear form (the representation portion of figure 2) is a trivial operation.
- 2. The linear forms of the graph can be directly represented without programming.

Notice (from figure 2) that an automated version of this KA methodology would coalesce the role of the analyst and the machine.

The knowledge acquisition methodology which we present is based on the following hypotheses:

- 1. Cogniting agents, including computers, understand the world by forming MMs.
- 2. MMs have a structure.
- 3. The structures of MMs can be modelled with conceptual graphs.
- 4. The operations on MMs can be modelled by using operations on conceptual graphs.

# 5 Knowledge Acquisition and Terminological Confusion

There is a good deal of terminological confusion surrounding many KA kinds of activities. Let us consider the word "knowledge", which in English is used as a mass noun, indicating something of substance and bulk. "Acquiring" knowledge suggests that it is a concrete entity that could be "acquired". Reality does not support the metaphor.

Contrary to the way that English speakers use the word "knowledge", it is obvious that the concept is as intuitively elusive and as difficult to define precisely as the concepts of information and entropy. Yet the successful handling of entropy in thermodynamics and the precise and fertile definition of the Shannon-Weaver concept of information should give us hope that the concept of knowledge will also yield to a fruitful and operationally meaningful definition. Unfortunately, that time has not yet arrived.

As a substitute, and following suggestions made in the literature [18,5], for the purposes of this paper, we define knowledge as the semantic content of mental models. Tying knowledge to a mental model implies that there is no knowledge without a knower. This is our intent. Knowledge is possessed by a knower only in so far as the MMs "contain" this knowledge.

. We assert that:

- 1. There is no knowledge without a knower.
- 2. There is no acquisition without an acquirer.
- 3. There is no acquisition without a source and, for us, the source is a cogniting agent.

Now let us consider systems analysis as a typical KA process. The role of analyst is played by the "systems analyst" and the role of informant is played by the "user".

Let us consider the problems in this scenario:

- The "user" may in fact have nothing to do with the use of the system under construction. The "user" may only act as a source of expertise in creating specifications for the future system. In what sense is this kind of a "user" a user?
- 2. The systems analyst thinks that he is analyzing a system, but this system does not exist, except perhaps in the mind of the analyst. In trying to analyze a non-existent system, certain distortions and inaccuracies are created as a side effect of the informant-analyst knowledge transfer.
- 3. The systems analyst would normally use the systems approach: He would interpret information received in terms of his view of the system which is usually an input-processoutput model. At times, the crucial issue is the structure of information which goes through the input and output processes and not so much the nature of the processes or the nature of the transformation which the information undergoes. While information flows are analyzed and carefully diagrammed on data flow diagrams, the structure of the information that flows is ignored. It is pushed over to database design.

Actually, a process of fact-finding does take place. Knowledge is being acquired by the analyst and this knowledge will be built into the computer system. If the systems analyst is analyzing anything, he is analyzing chaos. His job is create order out of chaos. Attempts to describe exactly what goes on during the process of systems analysis, conceptual database design, and requirements specification writing illustrate the confused state of the discourse about KA issues.

# 6 Providing Machines With Ontologies and Inventories

How does a machine know what is "out there in the world?" Write a program in the Fortran programming language which accepts as input, two integers, and returns, as output, their sum. Run the program and enter a real number and the letter k. What happens? The program aborts. Somewhere between the programmer and the user, somebody lied to the machine. The machine, when executing this program, "thinks" that there are only integer numbers "out there in the real world."

Does the computer, when executing this program have an ontology? Only implicitly. We define an *explicit ontology* to be a publicly examinable list of entity types. In a similar vein, we define an explicit inventory to be a publicly examinable list of instantiations listed in the explicit ontology.

Each cogniting agent has, as a minimum, an implicit ontology: man, animal, and machines all have implicit ontologies. Very few cogniting agents have explicit ontologies. The clearest example of an explicit ontology is a database schema which can be printed and is comprehensible to those who understand the data definition language. Databases also have explicit inventories in the form of population tables.

Fire insurance companies sometimes request that their policy holders complete an explicit ontology and an explicit inventory. Many accounting procedures (such as recording an inventory) can be viewed from the point of view of ontology and inventory. Philosophers are also interested in compiling ontologies although inventories are a foreign notion to them. Anthropologists, it could be argued, are also in the ontology and inventory business. They visit foreign cultures and as part of their field work, attempt to compile an explicit ontology. We believe that KA for systems development is fundamentally an anthropological activity.

How does a machine know what is "out there in the world?" This is the inverse of the situation confronting the anthropologist. The anthropologist seeks to find out; the machine wants to be told which entities are "out there." It is humans who are providing the machine with a MM. The analyst or programmer ultimately decides which entities in the world will be known to the machine. If the analyst or programmer misses a few, the system might fail at some point. This makes it imperative that the machine know about all of the relevant entities in the domain of discourse. In fact, the entities which the analyst-programmer team tells the machine about will constitute the domain of discourse.

What should be the basis for the ontology which we construct for the machine? The obvious choice would be to use "things in the world", things like employees, managers, employee numbers, departments, and department budgets.

Table 1:	Ta	ble	1:
----------	----	-----	----

1	system	subsystem	component
2	set	subset	element
3	domain of discourse	subdomain	entity
4	mental model	model fragment	concept
5	conceptual graph	subgraph	concept-node

There is a problem with this approach.<sup>6</sup> Both informants and analysts speak of nonexistent entities—things like systems. The system will only exist (hopefully) at some point in the future. And in what sense is an employee number a "thing in the world"? If we base the ontology on the "things in the world", we are restricting ourselves unnecessarily.

A second possibility is to use the names of entities as the basis for an ontology. This approach leads to three problems. First, names are also entities. Thus this approach requires distinguishing between lexical and non-lexical entities as is done in ENALIM diagrams with their distinction between LOTs (lexical object types) and NOLOTs (non-lexical object types) [9]. Second, the same name, as we all know, can refer to different entities. An ontology based on names is bound to lead to confusion. Third, many entities in the "real world" lack a name although they are part of the implicit ontology of an informant. Let us consider an example from systems analysis. In a manufacturing environment, some partially assembled structure must be moved from one department to another. The analyst wishes to describe this inter-departmental transfer and asks the informant for the name of the sub-assembly. The informant can provide no name: the subassembly is an unnamed object.

Restricting the ontology to "real world" entities will not work. Nor will a name based ontology work. We believe that the only basis for constructing an ontology is concepts. Concepts are components of MMs and can be connected to entities which are unnamed. Concepts can be named allowing the informant and analyst to discuss entities such as "system" for which no real world referent exist. While there are those who wish to keep their ontologies pure of "non-existing entities", we believe that a KA methodology should be more descriptive than normative.

To clarify our terminology, and to allow the reader a glimpse of our mental models we offer Table 1:

Items in a given column of table 1 are in the same taxonomic relationship to items in the other columns. Lines 1 and 2 give the hierarchical taxonomy of systems theory and set theory respectively. The ontology of the computer is restricted to line 5. The inventory of the computer consists of instantiated concept-nodes.

<sup>6</sup>The relationship between "data" and "reality" is discussed very thoroughly in William Kent's Data and Reality.



# 7 Knowledge Acquisition Through Text

If there is no knowledge without a knower, then knowledge is not to be found "in" a text. All the same, discourse, whether oral or written, does deservedly occupy a special, privileged place in KA.

How does a person go from words to knowledge? Words "create" knowledge by causing a cogniting agent to form new MMs or alter existing MMs. During discourse, the MMs of both informant and analyst are harmonized; the MMs are "brought together" using language as a technology of harmonization.<sup>7</sup>

From our point of view, KA reduces to Natural Language Understanding (NLU). We seek knowledge; we are given words. KA reduces to NLU: generating meaning for words. To generate a meaning for a word means to map a word into a concept where the concept can be represented as a conceptual graph.

John Sowa has provided a mechanism for connecting words and concepts. That mechanism is a conceptual lexicon which Sowa [18], in an example, organizes as a look-up table. For example, the lexeme "occupy" maps into the concepts [OCCUPY-ACT]. [OCCUPY-STATE], and [OCCUPY-ATTENTION]. These three concepts are illustrated by the following sentences:

- The enemy occupied the island with marines.
- Debbie occupied the office for the afternoon.
- Baird occupied the baby with computer games.

Some would say that the word "occupy" is being used in different senses. The "sense" of a word is a fairly crucial issue which will require further elaboration.

The sense of a word is defined in the *Concise Oxford English Dictionary* as: "meaning, way in which word etc. is to be understood." For the purpose of this paper, we define the "sense" of a lexeme "lex", where "lex" could be—and in fact typically would be—a phrasal lexeme, as the ordered pair

( "lex", [LEX n] )

where [LEX n] is the nth concept associated with the lexeme "lex". The sense predicate

S ( "lex", [LEX n] )

asserts that ( "lex". [LEX n] ) is a sense of the lexeme "lex".

It is doubtful that lexemes and concepts are associated in human memory as a simple look-up table. The process of "reminding" in the human mind is not very well understood.<sup>8</sup> Likewise, very little is known about the process that associates a lexeme with a concept.<sup>9</sup> However, from the point of view of KA, only the resulting concept [LEX n] is of significance and not the process that forms the pair

<sup>7</sup>We owe this key insight to Roy Hagman. See pages 104-107 of his Language, Life, and Behaviour.

<sup>&</sup>lt;sup>8</sup>All this talk of reminding reminds us of Roger Schank's book Dynamic Memory.

<sup>&</sup>lt;sup>9</sup>There is a body of psycholinguistic literature on this subject but there seems to be little agreement in the literature on how the lexeme-concept association process works.

( "lex", [LEX n] ).

in the mind of the informant.

Having acquired the concept [CONC], the analyst can describe [CONC] to the informant using both natural language and the formalism of conceptual graphs. Feedback from the informant further modifies the MM of the analyst. On the basis of the feedback, the analyst may create a new concept [CONC1]. This process can continue until both the analyst and the informant are satisfied or until the allotted time for the KA phase of the project has expired.

It should be noted that there are more sense relations than are defined in a dictionary. These "extra" senses are particular to individuals and hence idiosyncratic. Dealing with these complexities can be accomplished through a formalization of the meaning triangle [13].

# 8 An Interviewing Procedure for Knowledge Acquisition

While the interviewing process associated with a knowledge acquisition process may be quite extensive, in this paper we will restrict our attention to the particular task of compiling an ontology and inventory list for a particular domain of discourse. Statements by the informant would be about entities. A complete ontology list contains all the concept types and a complete inventory list contains all the instances that statements of the informant might refer to. We note that these statements are of no concern to us during the interview, except in so far as they give clues about the "existing" population of entities in the domain of discourse. Descriptive information is of use only in so far as it describes the conceptual components of the mental models of informant and analyst.

With these restrictions, the step-by-step procedure of interviewing is as follows:

Step 1. Establish a text. The text should be in a permanent, publicly accessible form. In practice, this would mean a written document, a voice tape, or a videotape.

Step 2. Select the contentives. Select all contentive lexemes, paying special attention to all the phrasal lexemes and subfragments of noun phrases.

Step 3. Produce a lexeme-to-concept mapping. The mapping is done either through a conceptual lexicon, if one exists, or through free association by the informant or analyst.

Step 4. Diagram the concept. Concepts that are semantic primitives are diagrammed as a single concept node containing the concept itself. For non-primitive concepts, the conceptual graph is as elaborate as necessary.

Step 5. Test the concepts. Elicit further comments from the informant on the appropriateness of the graphs. This step may produce more text which is to be cycled to Step 1 to produce a new version of the text.

Step 6. Model mental models. Model, using conceptual graphs, the relevant mental models of the informant to test the adequacy of the coverage. The main techniques are conceptual analysis and further dialogue with the informant. This step may produce only diagrams, in which case the interviewing procedure is complete, or more text, in which case cycle back to Step 1.

Clearly, the process may not terminate within a preset time limit. Cutoff criteria may have to be established for termination conditions.

The final product of this algorithm consists of two lists:

- 1. The Ontology List: a list of concepts, each defined by a conceptual graph. A conceptual graph consisting of a single concept node is a semantic primitive relative to this ontology.
- 2. The Inventory List: a list of instantiations of concept types relative to the particular domain of discourse.

It is assumed that there is a domain of discourse. The mental models of the informant are assumed to represent this domain of discourse. As a matter of practical fact, the mental models of the informant define the domain of discourse, if it is assumed that knowledge about the domain is only accessible through the informant. The analyst's task is to establish a set of mental models in his own mind which are coherent with the set of mental models of the informant. The analyst makes a publicly examinable record of his own mental models by drawing diagrams. If the diagrams do not adequately model the mental models of the informant, then a process of adjustment will have to take place. If we look upon language as a form of technology, then the process can be described as follows: the analyst and the informant are asked to synchronize mental models using the technology of language [4], assisted by a diagramming technique such as conceptual graphs.

### 9 Knowledge Acquisition – An Example

To illustrate our methodology we will run through a typical knowledge acquisition problem. In our example, the analyst is trying to find his way to a party and the informant provides directions. Admittedly, the matter is not one that requires a high degree of sophisticated expertise, but we have all been confronted by cases where the directions were so ineptly phrased that careful conceptual analysis was required to create order out of chaos.

To make the example more interesting, and to illustrate how two different ontologies and two different domains of discourse can be established for what, at an abstract level, are the "same" instructions, we shall give two versions of the text:

- 1. An object-oriented version.
- 2. A procedure-oriented version.

Giving the full analysis of the two texts would be too lengthy for this paper. Some selected examples will illustrate the techniques.

The following texts are two sets of instructions given by two informants to an analyst on how to get to a party:

1. Text 1: Object-oriented

"We live at 251 Elm St.. Elm is a north-south street. The nearest major intersection is George and Hunter. We are in the southwest quadrant."

Table 2:	
----------	--

we	live	251
Elm	St.	Elm St.
251 Elm St.	north	south
north-south	street	north-south street
nearest	major	intersection
major intersection	nearest major intersection	George
Hunter	southwest	quadrant
southwest quadrant		

### 2. Text 2: Procedure-oriented

"Where are you now? OK. Drive down George until you get to Hunter. Turn right. Turn left after the fourth block. That's Elm. Now go down six houses and we are on the right."

We believe that it will become apparent that the ontology and the domains of discourse are quite different for the two examples, in spite of the fact that they are talking about the "same" thing.

Let us step through the procedure described in the previous section. This is artificial because the two cogniting agents, the person who gave the instructions and the person who received the instructions, are not present. We have to invent their responses and we have to make some guesses about the mental models of both analyst and informant.

Let us take the object-oriented version first.

Step 1. Establish the text. This is done above.

Step 2 List the contentives.

Normally, this would be a very long list, but this text is relatively short. We produce this list in table 2.

Step 3. Produce a lexeme to concept mapping.

Looking at this list of contentives for Text 1, the analyst may decide to try to compile the first version of the ontology list. The concepts associated with the lexemes may turn out to be semantic primitives with no further definition, or may be defined by conceptual graphs which are constructed out of primitives and other non-primitive concepts. We show the lexeme to concept association in table 3.

Lexemes such as "intersection" and "quadrant" may be difficult to handle at this stage for the analyst. We shall illustrate some possible techniques for dealing with this later. We note that at this stage that the ontology and inventory lists for text 1 may look as shown in tables 4 and 5.

We note that conceptual graphs would be required to connect the concepts [COMPASS-DIRECTION] and [STREET-TYPE]. The analyst may also want to connect this objectoriented text to procedural information. In this case he may ask at some future point in the
Table 3	:	
---------	---	--

<b>Contentive Lexeme or Phrase</b>	Associated Concept
we	[PARTY-HOST]
live	[RESIDENCE]
251 Elm St.	[ADDRESS]
251	[HOUSE-NO]
north	[COMPASS-DIRECTION]
street	[STREET]
north-south street	[STREET-TYPE]
nearest	[NEAR]
George	[STREET:George]
southwest	[COMPASS-DIRECTION]

Table 4:

Text 1 Ontology List: version 1
[PARTY-HOST]
[RESIDENCE]
[ADDRESS]
[HOUSE-NO]
[COMPASS-DIRECTION]
[STREET]
[STREET-TYPE]
[NEAR]

Table 5:

Text 1 Inventory List by Concept Type: version 1
[ADDRESS: 251 Elm St. ]
[HOUSE-NO: 251]
[STREET-TYPE: north-south street ]
[COMPASS-DIRECTION: north, south, southwest]
[STREET: Elm, Hunter, George]

Table 6:

where	you	now
OK	drive	drive down
George	get	get to
Hunter	turn	turn right
turn left	block	fourth
fourth block	after the fourth block	Elm
go	go down	six
houses	six houses	we
right	left	on the right

process: "Yes, I hear what you are saying, but how do I actually get to your house?" At this point more text will be generated.

To contrast, let us look at the first two steps for Text 2.

Step 1. Establish the text.

This is done above.

Step 2. List the contentives.

The contentives are shown in table 6.

We note that Text 2 is not only procedure-oriented, but also more colloquial. Colloquial language is usually ill-formed and idiosyncratic, and also makes references to entities which require a great deal of background knowledge to understand. But the analyst can still try to associate concepts with lexemes, although the process may require further information from the informant.

Now let us return to the processing of Text 1.

Step 3. Produce a lexeme-to-concept mapping (cont.)

The analyst may now want to tackle some of the more difficult concepts. Let us say that the analyst starts with the lexeme "intersection". Relying on his own mental models, and using his own "reminding" processes, the analyst may come up with the following natural language phrasings to capture the various sense of the lexeme "intersection":

- 1. "a place where one can pass from one street to another" (captures the concept [INTERCHANGE])
- "a place where two streets meet" (captures the concepts of [T-JUNCTION], as well as [FORK-IN-THE-ROAD])
- "a place where two streets meet at right angles creating four quadrants" (capturing the concept of [CROSSROADS])

The analyst may not be satisfied with any of these natural language phrasings of the sense for "intersection" and may, if there is no conceptual lexicon available, consult dictionaries and other sources for further information.

15

Suppose the analyst decides that Sense 2 and Sense 3 are the best candidates. For the sake of version control, he may not add the statements of Sense 2 and Sense 3 to the text as yet.

Step 4. Diagram the concept using a conceptual graph.

The analyst decides initially that for [INTERSECTION2] the concept will be [JUNCTION] and that for [INTERSECTION3] the concept will be [CROSSROADS].

He draws conceptual graphs for [INTERSECTION1] and [INTERSECTION2] and shows them to the informant.

Step 5. Elicit comments from the informant.

The informant expresses his opinion that [CROSSROADS] is the concept that best matches his mental models, i.e. that [CROSSROADS] is what he "had in mind" when he was giving instructions. He volunteers the following extra prose:

Well, think of analytic geometry. The two axes intersect, dividing the plane into four quadrants. The four city blocks around an intersection, provided it is an intersection formed by two streets intersecting at right angles, are similar to the four quadrants. You remember how they are labelled: the first quadrant, the second quadrant, the third quadrant, and the fourth quadrant. Well, when you look at a compass rose, you see the same thing. Obviously, the southwest quadrant according to the compass rose is the third quadrant of analytic geometry.

This step provided more text, namely the natural language description for the sense predicate S ( "intersection". [INTERSECTION3]) above, plus the extra information provided by the informant. This text should be added to the original version to produce Version 2 of the text. Now the analyst can enlarge the list of contentives above to produce Version 2 of the contentives list. Then, he may update the ontology list by adding the conceptual graph for [INTERSECTION3] to the list.

At the end of this iteration from Step 1 to Step 5 we have

- Text Version 2
- Contentives list Version 2
- Ontology list Version 2

#### Step 6. Model through diagrams the mental models of the informant.

Making an appropriate decision on the depth of the analysis, the analyst could proceed to elicit additional information about the mental models of the informant. Conceptual analysis is described by Aaron Sloman [17] and John Sowa [18], as well as by Christopher Riesbeck in [15]. The process may not converge, unless decisions are taken by informant and analyst on what does, or does not, belong to the domain of discourse.

The final product is an ontology list which contains all the concepts that are semantic primitives, as well as the concepts that are defined by more complex conceptual graphs. The inventory list is assembled on the basis of the contentives list. Each item in the inventory is

an instantiation of a concept type. Matching up items on the contentives list with concept types is based on information from the informant.

For example, the concept [STREET] may be accepted as a semantic primitive, and the contentives "Elm", "George", and "Hunter" as instances of [STREET]. This could be recorded as a population table, or in the linear notation of conceptual graphs, as

[STREET:Elm, George, Hunter]

## 10 Conclusion

The necessity of relying on an outside expert as the source of the knowledge in the creation of knowledge-based systems forces us to state more clearly than ever before exactly what steps must be carried out during the KA process. Writing such a "procedures manual" for people engaged in person-to-person KA work is a prerequisite for creating the architecture and specifications for an automated knowledge acquisition methodology. We are still a long way from an acquisition methodology which can be automated: that future seems distant. We offer our methodology as a workable approach for performing person-to-person knowledge acquisition today, in the present.

### References

- Brachman, Ronald and Levesque, Hector, eds. Readings in Knowledge Representation, Los Altos, CA: Morgan Kaufmann, 1985.
- [2] Clancey, W.J.
  "Review of J.F. Sowa's Conceptual Structures", Artificial Intelligence, 27 (1985), pp. 113-128.
- [3] Fargues, Jean, Marie-Claude Landau, Anne Dugourd, Laurent Catach.
  "Conceptual Graphs for Semantics and Knowledge Processing", IBM Journal of Research and Development, Volume 30, Number 1, January 1986, pp. 70-79.
- [4] Hagman, Roy S.

Language, Life, and Human Nature, Current Inquiry into Language and Human Communication 41, Edmonton: Linguistic Research, 1982.

- [5] Johnson-Laird, Philip Mental Models, Cambridge, Ma.: Harvard University Press, 1983.
- [6] Kent, William Data and Reality: Basic Assumptions in Data Processing Reconsidered, New York: Elsevier Science Publishers, 1978.
- [7] Laing, R.D. Self and Others, Harmondsworth, England: Penguin Books, 1961.
- [8] McCracken, Daniel D. and Jackson Michael A.
  "Life Cycle Concept Considered Harmful", Software Engineering Notes, Volume 7, No. 2, April 1982, pp. 29-32.
- [9] Olle, T.W., Sol, H.G. and Verrijn-Stuart, A.A. (eds.) Information Systems Design Methodologies: Proceedings of the IFIP WG 8.1 Working Conference, Noordwijkerhourt, The Netherlands, 10-14 May (1982), Amsterdam: North Holland, 1983.
- [10] Plantinga, Edwin P.O.
  "Towards Automated Methods for Developing User Interfaces", Technical Report 109, Department of Computer Science, University of Western Ontario, 1983.
- [11] Plantinga, Edwin P.O.
  "Who Decides What Metaphors Mean?", Proceedings of the Conference on Computing and the Humanities—Today's Research, Tomorrow's Teaching, Toronto, April 1986, pp. 194-204.
- [12] Quine, W.V.O. Word and Object, Cambridge, Ma.: MIT Press, 1960.

18

- [13] Regoczei, Stephen "Formalizing the Meaning Triangle", Trent University Technical Report, To appear.
- [14] Regoczei, Stephen "Can Computers—or, for That Matter, People—Understand Natural Language, and if Not, Why Not?—the Impact of the Cognitive Revolution", To appear.
- [15] Schank, Roger C., (ed.) Conceptual Information Processing, Amsterdam: North Holland, 1975.
- [16] Schank, Roger C. Dynamic Memory: A Theory of Reminding and Learning in Computers and People, Cambridge: Cambridge University Press, 1982.
- [17] Sloman, Aaron

The Computer Revolution in Philosophy: Philosophy, Science, and Models of Mind, Harvester Studies in Cognitive Science, Atlantic Highlands, New Jersey: Humanities Press, 1978.

[18] Sowa, John F.

Conceptual Structures: Information Processing in Mind and Machine, Reading, Ma.: Addison-Wesley, 1984.

[19] Sowa, John F., and Way, Eileen
 "Implementing a Semantic Interpreter Using Conceptual Graphs", *IBM Journal of Research and Development*, Volume 30, Number 1, January 1986, pp. 57-69.

3879

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

### TECHNIQUES FOR KNOWLEDGE ACQUISITION AND TRANSFER

### Mildred L G Shaw & Brian R Gaines Department of Computer Science, University of Calgary Calgary, Alberta, Canada T2N 1N4

Abstract: This paper gives a state-of-the-art report on the use of techniques based on personal construct psychology to automate knowledge engineering for expert systems. It presents the concept of knowledge support systems as interactive knowledge engineering tools, states the design criteria for such systems, and outlines the structure and key components of KSS1 and its KITTEN implementation. KSS1 includes tools for interactive repertory grid elicitation and entailment analysis that have been widely used for rapid prototyping of industrial expert systems. It also includes tools for text analysis, behavioral analysis and schema analysis, that offer complementary and alternative approaches to knowledge acquisition. The KITTEN implementation integrates these tools around a common database with utilities designed to give multiple perspectives on the knowledge base.

### **KNOWLEDGE SUPPORT SYSTEMS**

Problems of knowledge engineering have been recognized since the early days of expert systems. It was possible that knowledge engineering might develop as a profession on a par with systems analysis and programming, and that an initial shortage of skilled knowledge engineers would cause problems to be overcome eventually be as the profession developed. However, this scenario now appears less and less likely. There is certainly a shortage of knowledge engineers and problems in developing applications, but doubts have been cast on the notion that human labor is the appropriate solution to the knowledge engineering problem:

- The decline in costs of both hardware and software support for expert systems has brought the technology into a mass-market situation far more rapidly than originally envisioned;
- This has lead to a growth in demand for expert systems that is proceeding far more rapidly than the growth in supply of trained and experienced knowledge engineers;
- The declining costs of expert system technology are also making the expense of human labour in tailoring the technology for particular applications appear to be the dominating constraint and an excessive cost;
- A move towards a labor-intensive activity such as knowledge engineering is contrary to all trends in industry;
- In particular it is contrary to the trend towards automatic programming techniques in the computing industry;
- The role of the knowledge engineer as an intermediary between the expert and the technology is being questioned not only on cost grounds but also in relation to its effectiveness—knowledge may be lost through the intermediary and the expert's lack of knowledge of the technology may be less of a detriment than the knowledge engineer's lack of domain knowledge.

The considerations of the previous section have heightened interest in the possibility of providing knowledge support systems (KSSs) to automate knowledge engineering as a process of direct interaction between domain experts and the computer. Gaines & Shaw (1980) proposed that personal construct psychology (Kelly 1955, Shaw 1980) could provide foundations for expert systems, particularly in systems that combined interactivity with database access and expert advice to provide decision support, and gave examples of algorithms and programs that extracted entailment rules from repertory grid data. Shaw & Gaines (1983) reported on further enhancements of these techniques and a preliminary experiment to validate them empirically as a knowledge engineering technique for priming expert systems. This work led to industrial studies of the methodology applied to the development of expert systems: Boeing Computer Services (Boose 1984, 1985, 1986) and Lockheed Software Technology Center (Wahl 1986) have reported success in applications; and validation has been reported in a statistics domain (Gammack & Young 1985).

This paper gives a state-of-the-art report on the use of techniques based on personal construct psychology to automate knowledge engineering for expert systems. It is based on four areas of advance since the previous paper:

- Improved techniques for the derivation of rules from repertory grid data which give: a natural knowledge representation for uncertain data combining fuzzy and probabilistic logics; and an information-theoretic measure of the significance of a derived rule (Gaines & Shaw 1986a);
- Widespread applications experience in prototyping expert systems using the methodology (Boose 1985, Gaines & Shaw 1986b);
- Improved interactive techniques for on-line knowledge engineering from groups of domain experts interacting through a computer network (Shaw 1986, Shaw & Chang 1986);
- KSS1, and its KITTEN implementation, a knowledge engineering workbench that provides next generation KSS facilities including textual analysis, induction of models

from behavior, multi-level and multi-expert repertory grid elicitation, and hierarchical construct laddering, to automate knowledge engineering for a wide range of problem domains.

### KNOWLEDGE SUPPORT SYSTEM DESIGN CONSIDERATIONS

Knowledge engineering may be seen in very broad terms as:

the acquisition, elicitation, structuring and encoding of knowledge for application in inferential, goal-directed, explanatory, decision and action support systems.

Knowledge support systems have even broader scope, encompassing both aids to knowledge engineering and support of human knowledge processes—in the long term the division between knowledge engineering tools and expert system shells will break down, and integrated systems will be necessary. The general requirements for a KSS are:

- 1. The KSS tools should be domain independent;
- 2. The KSS tools should be directly applicable by experts without intermediaries;
- 3. The KSS tools should be able to access a diversity of knowledge sources including text, interviews with experts, and observations of expert behavior;
- 4. The KSS system should be able to encompass a diversity of perspectives including partial or contradictory input from different experts;
- 5. The KSS system should be able to encompass a diversity of forms of knowledge and relationships between knowledge;
- 6. The KSS system should be able to present knowledge from a diversity of sources with clarity as to its derivation, consequences and structural relations;
- 7. Users of the KSS should be able to apply the knowledge in a variety of familiar domains and freely experiment with its implications;
- 8. The KSS should make provision for validation studies;
- 9. As much of the operation of the KSS as possible should be founded on well-developed and explicit theories of knowledge acquisition, elicitation and representation;
- 10. As the overall KSS develops it should converge to an integrated system.

All of these requirements are subject to caveats—some domain dependence may be appropriate for efficiency in specific KSSs—some human intervention may be helpful or necessary when an expert is using a KSS—and so on. However, the broad design goals stated capture the key issues in KSS design currently.

The PLANET system for repertory grid elicitation and analysis (Shaw 1980, 1982, Shaw & Gaines 1986b,c) is a primitive KSS satisfying requirements 1 and 2 for domain independence and direct use. Its foundations in personal construct psychology, which itself has strong systemic and cognitive science foundations (Gaines & Shaw 1981, Shaw & Gaines 1986a), are attractive in terms of requirement 9. Boose (1985) in evaluating ETS has noted the limitations of basic repertory grid techniques in terms of requirement 5—that the methodology is better suited for analysis than for synthesis problems, for example, debugging, diagnosis, interpretation and classification rather than design and planning, and that it is difficult to apply to deep causal knowledge or strategic knowledge—and is attempting to overcome these use grid hierarchies in NeoETS (Bradshaw & Boose 1986). The TEIRESIAS extension to MYCIN is an early form of KSS providing debugging support for an expert system using basic analogical reasoning (Davis & Lenat 1982). The development of KSSs has become a major area of activity recently, for example, MORE (Kahn, Nowlan & McDermott 1985), SALT (Marcus, McDermott & Wang 1985), SEAR (van de Brug, Bachant & McDermott 1985), and MOLE (Eshelman & McDermott 1986).

The following section describes our work on KSS1, a knowledge support system that draws on many concepts and techniques for knowledge engineering to begin to encompass requirements 3 through 8, while attempting to satisfy 9 by relating them all through personal construct psychology, and 10 by building a workbench of tools around a common database.

### KSS1: A KNOWLEDGE SUPPORT SYSTEM

Figure 1 shows the structure of KSS1 (in its workstation implementation termed KITTEN: Knowledge Initiation & Transfer Tools for Experts and Novices). KSS1 consists of a: knowledge base; various analytical tools for building and transforming the knowledge base; and a number of conversational tools for interacting with the knowledge base. The KITTEN implementation is written in Pascal and currently runs on a coupled IBM AT and Apple Macintosh to combine processing power and interactivity.



Figure 1 KSS1—KITTEN— Knowledge Initiation and Transfer Tools for Experts and Novices

The KSS1 structure is best understood by following sequences of activity that lead to the generation of a rule base and its loading into an ES shell.

A typical sequence is text input followed by text analysis through TEXAN which clusters associated words leading to a schema from which the expert can select related elements and initial constructs with which to commence grid elicitation. The resultant grids are analyzed by ENTAIL which induces the underlying knowledge structure as production rules that can be loaded directly into an ES shell (Gaines & Shaw 1986a).

An alternative route is to monitor the expert's behavior through a verbal protocol giving information used and decisions resulting and analyze this through ATOM which induces structure from behavior and again generates production rules (Gaines 1977).

These two routes can be combined. KSS1 attempts to make each stage as explicit as possible, and, in particular, to make the rule base accessible as natural textual statements rather than technical production rules.

The ES shell being used in KITTEN currently is Nexpert (Roy 1986) which gives a variety of textual and graphical presentations of the rule base enabling the expert to see the impact of different fragments of knowledge.

The group problem-solving component of KSS1 is particularly important because it goes beyond the stereotype of an "expert" and "users", and allows the system to be used to support an interactive community in their acquisition and transfer of knowledge and mutual understanding. The SOCIO analysis allows members of a community to explore their agreement and understanding with other members, and to make overt the knowledge network involved (Shaw 1980, 1981).

The KITTEN implementation is an initial KSS1 prototype offering a workbench with minimal integration of the knowledge base, but each of tools has already proven effective, and their combination is proving very powerful in stimulating experts to think of the knowledge externalization process from a number of different perspectives.

The following sections describe and illustrate some of the tools.

#### ENTAIL: ENTAILMENT ANALYSIS

PLANET and ETS access the expert's personal construct system by interactively eliciting a repertory grid of constructs classifying elements characterizing to part of the domain of expertise. A repertory grid, such as that of Figure 2, may be viewed as a component of a database in entity-antribute form with elements as entities, constructs as attributes and ratings of elements on poles of constructs as values.

The entailment analysis of a repertory grid treats each pole of a construct as a fuzzy predicate to which the elements have degrees of membership given by their ratings, and induces the logical implications between these predicates. The original ENTAIL program produced all entailments consistent with the grid and allowed the expert to prune any that seemed spurious before using them as inference rules in an expert system. ENTAIL II rank orders entailments in terms of the uncertainty reduction they induce in the distribution of the data, and hence tends to reject spurious entailments (Gaines & Shaw 1986a).

Figure 3 is an ENTAIL II analysis of the grid of Figure 2. The entailments are shown with three values in the range from 0 to 1: first, the truth value of the hypothesis; second, the probability of the hypothesis being true; and third, the information content (uncertainty reduction generated) of asserting the hypothesis. For example,  $L1 \rightarrow L9$  has a truth value of 0.80, a probability of 1.00, and an information content of 0.29. The information content measures the significance of the hypothesis and is used to ensure that trivial entailments consistent with the data are pruned.

The data of Figure 2 may be regarded as that of an expert on staff appraisal concerned with deriving his overall rating (construct 13) from behavioral assessments such as intelligent and creative. The ENTAIL analysis of Figure 3 shows that L1, L4, L6, L9, L10 and L12 imply L13, that intelligent, creative, reliable and professional self-starters requiring little supervision receive a high overall rating, whereas R2, R4, R5, R6, R9 and R12 imply R13, that being unwilling, less motivated, not so reliable, less professional, needing supervision and needing a push leads to a low overall rating.

\*1 2 3 4 5 6 7 8 9 10 \* intelligent 1 \* 1 1 4 5 3 3 5 2 3 5 \* 1 dim 2 4 з willing 2 1 5 1 1 4 1 2 \* 2 unwilling new boy 3 2 3 5 4 4 3 \* 3 old sweats 1 4 4 1 little supervision reqd 4 3 1 4 5 2 1 5 2 2 3 4 need supervision motivated 5 1 1 4 5 2 2 5 3 3 2 5 less motivated 2 2 5 1 1 reliable 6 3 5 1 2 3 6 not so reliable \* 7 4 5 2 2 3 1 5 5 7 abrasive mild 3 4 ideas men 8 1 5 4 2 3 1 3 4 8 staid 5 3 self starters 9 2 1 5 1 3 5 4 5 9 need a push creative 10 1 5 5 2 3 3 5 10 non-creative 1 4 4 helpful 11 \* 4 34 2 3 5 1 4 5 5 \* 11 unhelpful professional 12 \* 1 2 3 3 2 1 5 24 4 \* 12 less professional overall rating high 13 \* 2 1 3 4 1 2 5 2 3 4 \* 13 overall rating low messers 14 \* 2 2 5 4 3 5 1 5 3 1 \* 14 tidy \* \* \* S10 \* \* \* S9 \* \* \* \* \* \* \* \* **S**8 \* S7 \* \* \* \* S6 \* \* \* S5 \* \* \* S4 \* \* **S**3 \* S2 **S**1

#### Figure 2 Repertory grid elicited on staff appraisal

Figure 4 shows Nexpert in operation loaded with the entailments of Figure 3. Interaction with Nexpert enables the expert to see the derived rules in action. He can determine their consequences with test data, analyze new hypothetical cases, and see the inter-relations between rules presented graphically. The logging and explanation facilities of Nexpert enable him to track down spurious inferences that may arise with the rules derived by ENTAIL, or proper inferences that are missing. He can then edit the rules and test the revized system using Nexpert's facilities.

In some applications the data in the grid about elements may also be relevant and can be loaded in as a set of rules based on the ratings of the elements on the constructs (Boose 1984), for example, that S1 is intelligent. In other applications the relations between the constructs of different experts may be significant and can be derived by combining multiple grids (Bradshaw & Boose 1986).

Entail	Truth	Prob.	Inf.	(Cutoff 0.17) Implication Usually
$L 1 \rightarrow L 9$	0.80	1.00	0.29	intelligent→self starters
$L 9 \rightarrow L13$	1.00	1.00	0.29	self starters→overall rating high
$R 9 \rightarrow R 1$	0.80	1.00	0.28	need a push→dim
$L10 \rightarrow L 8$	1.00	1.00	0.28	creative→ideas men
$L 1 \rightarrow L10$	0.80	1.00	0.26	intelligent→creative
$R \ 8 \rightarrow R10$	1.00	1.00	0.26	staid→non-creative
$L10 \rightarrow L 9$	0.80	1.00	0.26	creative→self starters
$R13 \rightarrow R 6$	0.80	1.00	0.26	overall rating low→not so reliable
$L 9 \rightarrow L10$	0.80	1.00	0.24	self starters→creative
$R10 \rightarrow R 1$	0.80	1.00	0.24	non-creative→dim
$L10 \rightarrow L 1$	0.80	1.00	0.23	creative→intelligent
$R13 \rightarrow R 9$	1.00	1.00	0.23	overall rating low->need a push
$R 4 \rightarrow R13$	0.80	1.00	0.22	need supervision→overall rating low
$R 5 \rightarrow R 4$	0.80	1.00	0.22	less motivated → need supervision
$R 5 \rightarrow R13$	0.80	1.00	0.22	less motivated—overall rating low
$R 9 \rightarrow R10$	0.80	1.00	0.22	need a push->non-creative
$L 1 \rightarrow L 3$	0.80	1.00	0.21	intelligent->new boy
$L 6 \rightarrow L13$	0.80	1.00	0.21	reliable->overall rating high
$R10 \rightarrow R 9$	0.80	1.00	0.20	non-creative-need a push
$R 1 \rightarrow R 6$	0.60	1.00	0.19	dim→not so reliable
$R 1 \rightarrow R10$	0.80	1.00	0.19	dim→non-creative
$R 9 \rightarrow R 4$	0.60	1.00	0.19	need a push-need supervision
$R 9 \rightarrow R12$	0.60	1.00	0.19	need a push→less professional
$R 9 \rightarrow R13$	0.60	1.00	0.19	need a push->overall rating low
$R12 \rightarrow R13$	0.80	1.00	0.19	less professional→overall rating low
$R13 \rightarrow R 4$	0.80	1.00	0.19	overall rating low-need supervision
$R13 \rightarrow R12$	0.80	1.00	0.19	overall rating low→less professional
$L 4 \rightarrow L 5$	0.80	1.00	0.18	little supervision regd-motivated
$L 4 \rightarrow L 9$	0.60	1.00	0.18	little supervision regd-self starters
$R   \rightarrow R  4$	0.80	1.00	0.18	not so reliable-need supervision
$R 6 \rightarrow R13$	0.80	1.00	0.18	not so reliable→overall rating low
$L12 \rightarrow L 9$	0.60	1.00	0.18	professional→self starters
$L13 \rightarrow L4$	0.80	1.00	0.18	overall rating high->little supervision read
$L13 \rightarrow L 5$	0.80	1.00	0.18	overall rating high-motivated
$L13 \rightarrow L 9$	0.60	1.00	0.18	overall rating high-self starters
$L 1 \rightarrow L 8$	0.80	1.00	0.17	intelligent→ideas men

۰.

Figure 3 ENTAIL analysis of repertory grid on staff appraisal



Figure 4 Inference rules derived by ENTAIL in Nexpert expert system shell

### TEXAN: TEXT ANALYSIS

Repertory grid techniques depend on eliciting elements and constructs from experts that are representative of a domain and comprehensive in their classification. The interative elicitation program PEGASUS in PLANET uses online analysis of the grid to feed back comments to the expert which stimulate the addition of elements and constructs to achieve comprehensiveness (Shaw 1980). However, this structural feedback is only applicable when a grid has been partially completed and the initial selection of elements has had no computer-based support.

TEXAN is a text analysis program designed to pump-prime the grid elicitation process when a manual or text book is available that the expert regards as having reasonable coverage of the domain. It uses techniques that were originally designed to map subject matter concepts against student concepts in computer-managed instruction systems (Smith 1976). The text is fully indexed by all non-noise words grouped by their stems, and a coupling matrix of word associations is calculated using a simple distance-in-text measure. The high-frequency associations in the text are clustered and presented to the expert as a prototypical schema for the subject area which he can edit for spurious words and associations, and then use to suggest knowledge islands and associated elements and constructs.

Figure 5 shows a TEXAN clustering of an evaluation study of data logging, analysis and presentation methodologies for human performance evaluation in complex systems (Gaines & Moray 1985). Figure 6 shows an independent mapping of the main knowledge islands for an expert system design based on the analyzed report (Gaines 1986). The TEXAN analysis was done some time after the production of Figure 6, and the shading of Figure 5 shows the relationship of some of the groupings in the schema with the knowledge islands. There is not a one-to-one correspondence but this, and similar analyses, show that basic text analysis can focus attention on salient features of the domain and pump-prime the knowledge elicitation process.

In the long term more sophisticated text analysis techniques may be used to derive knowledge from text without human intervention. However, for many domains the knowledge is not yet that explicit and pump-priming of elicitation from experts will remain a significant requirement.

#### ANALOG: SCHEMA ANALYSIS

The groupings of Figures 5 and 6 when combined with the construct classifications of repertory grids as in Figure 2 may be viewed as schema structuring a knowledge domain. ANALOG is a program that maps schema to schema based on their structure without regard to content. It is based on a theory of analogy that explicates analogies as pullbacks of faithful functors between categories (Gaines & Shaw 1982) and generates maximal sub-graph isomorphisms between two classificatory data structures. It may be regarded as a generalization of the copy-edit process being used in the encoding of commonsense knowledge in CYC (Lenat, Prakash & Shepherd 1986). ANALOG produces meaningful results on artificial examples and grids in related domains. It will also find meaningless analogies between unrelated domains which cannot be rejected by information-theoretic statistical procedures such as those used in ENTAIL and ATOM. It seems likely that effective application of ANALOG depends on the expert pump-priming the matching with known or hypothesized relations and the program extending these rather than attempting to generate them completely.

#### ATOM: BEHAVIOR ANALYSIS

Michalski and Chilausky (1980) have demonstrated that inductive modeling of an expert's behavior may produce effective rules when those elicited by interview techniques are clearly inadequate. ATOM is an algorithm for inducing the structure of a system from its behavior using a search over a model space ordered by complexity and goodness of fit. As



۰,

Figure 5 TEXAN clustering of word associations from text with annotation showing knowledge islands





in ENTAIL, models are evaluated in terms of the uncertainty reduction induced by the model in the distribution of the modeled behavior (Gaines 1976, 1977, 1979). A version of ATOM has been incorporated in KITTEN that takes a set of sequences of arbitrary symbolic data and generates a set of production rules that will reconstruct it. These can be loaded into the ES shell to give a simulator of the behavioral system. This has proved effective with inter-personal interaction data such as that analyzed by Mulhall (1977) and interactively elicited by Stevens (1985).

#### SOCIO: MUTUAL AGREEMENT AND UNDERSTANDING ANALYSIS

This paper has already emphasized the need for knowledge elicitation methodologies to cope with a group of experts as well as the individual. Much expertise only resides within the social context of cooperating individuals and requires elicitation across the group. The SOCIO analysis program supports group elicitation techniques in which the construct systems of a number of users are compared. Grids are elicited separately but then exchanged in two ways: a user can place elements on a colleague's constructs from his own point of view, and the analysis system then allows him to explore their agreement; or he can attempt to place them from his colleague's point of view and hence explore his understanding.

The SOCIO program has been used to develop a Participant Construct System, PCS, which supports multiple interacting users in group problem solving activities through terminals on a network of Macintoshes (Shaw 1986). It follows the paradigm suggested by Chang (1986) of a participant system in which computer-based communication between multiple users is essential to the performance of their tasks, rather than merely incidental to their use of a timeshared computer or computer network. PCS supports its users in:

- seeing the relationship of their points of view to those of others;
- exploring differing terminology for the same constructs;
- becoming aware of differing constructs having the same terminology;
- extending their own construct systems with those of others;
- providing others with constructs they have found valuable;
- exploring a problem-solving domain using the full group resources.

Figure 7 shows a construct being elicited using the natural click-and-drag techniques of the window/mouse interaction rather than numeric rating scale of Figure 2.



Figure 7 Rating elements on a construct spatially using click and drag

Figure 8 shows the natural representation of the construct match screen supporting element elicitation.



Figure 8 Eliciting an element by breaking a construct match

Figure 9 shows the presentation of agreement when one participant has made his own assignment of elements to constructs elicited from another participant.

	File Edit Exchange	Compare	
E		Agreement with Fred	
	lively works long hours knows more than I do been here a long time sticks to the rules quick decision maker gets things done	dull finishes early not very knowledgeable new to job skates round rules takes forever to decide doesn't do much	
(	Cancel		
L.			



### CONCLUSIONS: STEPS TOWARD AN INTEGRATED KSS

This paper has presented the concept of knowledge support systems as interactive knowledge engineering tools, stated the design criteria for such systems, and outlined the structure and key components of KSS1 and its KITTEN implementation. KSS1 consists of a set of knowledge engineering tools, some of which already have track records of successful use in knowledge acquisition studies. In developing KITTEN the integrity of each of these tools has been preserved, enabling each to be utilized effectively in a standalone mode. However, the first steps have also been made towards an integrated knowledge support system by building the tools around a common database, providing access to the same data in each of its intermediate forms, and providing conversion utilities between different data forms.

The objective of integrating the tools has raised a number of new and significant questions. ENTAIL transforms a repertory grid to a set of production rules—is it possible, and useful, to convert production rules to a repertory grid? Technically the result is a possible world of grids that might have generated the rules, and the capability does prove useful, particularly given the other grid analysis tools available in KITTEN. Similar considerations apply to the transformations between other forms of knowledge representation. The next generation of knowledge support tools will be increasingly flexible in handling all aspects of knowledge acquisition, representation, processing and presentation. They will not be optimized with a particular knowledge representation, uncertainty calculation, inference mechanism, and so on, that are in some sense right. Rather they will provide a wide range of perspectives on the knowledge base, preserving source data and chains of derivative processes, so that users can freely explore the knowledge or follow a very specific path according to their choices and needs.

#### ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the National Sciences and Engineering Research Council of Canada. The PCS implementation is being carried out by Alberta Research Council. We are grateful to John Boose and Jeff Bradshaw of Boeing AI Center, and Ernie Chang of Alberta Research Council for stimulating discussions relating to knowledge support systems.

#### REFERENCES

Boose, J.H. (1984). Personal construct theory and the transfer of human expertise. Proceedings AAAI-84, 27-33. California: American Association for Artificial Intelligence.

Boose, J.H. (1985). A knowledge acquisition program for expert systems based on personal construct psychology. International Journal of Man-Machine Studies 20(1), 21-43 (January).

Boose, J.H. (1986). Rapid acquisition and combination of knowledge from multiple experts in the same domain. Future Computing Systems, In press.

Bradshaw, J.M. & Boose, J.H. (1986). NeoETS. Proceedings of North American Personal Construct Network Second Biennial Conference. pp. 27-41. University of Calgary: Department of Computer Science (June).

Chang, E. (1986). Participant systems. Future Computing Systems, in press.

- Davis, R. & Lenat, D.B. (1982). Knowledge-Based Systems in Artificial Intelligence. New York: McGraw-Hill.
- Eshelman, L. & McDermott, J. (1986). MOLE: a knowledge acquisition tool that uses its head. Technical Report. Carnegie-Mellon University: Department of Computer Science.

Gaines, B.R. (1976). Behaviour/structure transformations under uncertainty. International Journal of Man-Machine Studies, 8(3), 337-365 (May).

Gaines, B.R. (1977). System identification, approximation and complexity. International Journal of General Systems, 3, 145-174.

- Gaines, B.R. (1979). Sequential fuzzy system identification. International Journal of Fuzzy Sets and Systems, 2(1), 15-24 (January).
- Gaines, B.R. (1986). Development of performance measures for computer-based manmachine interfaces: Application to previous SHINMACS evaluation. Technical Report DCIEM-PER-SUP:MAR86.
- Gaines, B.R. & Moray, N. (1985). Development of performance measures for computerbased man-machine interfaces. Technical Report DCIEM-PER-FIN:JUL85.
- Gaines, B.R. & Shaw, M.L.G. (1980). New directions in the analysis and interactive elicitation of personal construct systems. International Journal of Man-Machine Studies 13(1) 81-116 (July).
- Gaines, B.R. & Shaw, M.L.G. (1981). A programme for the development of a systems methodology of knowledge and action. Reckmeyer, W.J., Ed. General Systems Research and Design: Precursors and Futures. pp. 255-264. Society for General Systems Research (January).
- Gaines, B.R. & Shaw, M.L.G. (1982). Analysing analogy. Trappl, R., Ricciardi, L. & Pask, G., Eds. **Progress in Cybernetics and Systems Research**. Vol. IX, pp. 379-386. Washington: Hemisphere.
- Gaines, B.R. & Shaw, M.L.G. (1986a). Induction of inference rules for expert systems. Fuzzy Sets and Systems, 8(3), 315-328 (April).
- Gaines, B.R. & Shaw, M.L.G. (1986b). Knowledge engineering for an FMS advisory system. Lenz, J.E. (Ed.) Proceedings of the Second International Conference on Simulation in Manufacturing: AMS'86. 51-61. Bedford, UK: IFS Conferences (June).
- Gammack, J.G. & Young, R.M. (1985). Psychological techniques for eliciting expert knowledge. Bramer, M., Ed. Research and Development in Expert Systems, pp. 105-116. Cambridge University Press.
- Kahn, G., Nowlan, S. & McDermott, J. (1985). MORE: an intelligent knowledge acquisition tool. Proceedings of the Ninth International Joint Conference on Artificial Intelligence. pp. 581-584. California: Morgan Kaufmann.
- Kelly, G.A. (1955). The Psychology of Personal Constructs. New York: Norton.
- Lenat, D., Prakash, M. & Shepherd, M. (1986). CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. AI Magazine 6(4), 65-85.
- Marcus, S., McDermott, J. & Wang, T. (1985). Knowledge acquisition for constructive systems. Proceedings of the Ninth International Joint Conference on Artificial Intelligence. pp. 637-639. California: Morgan Kaufmann.
- Michalski, R.S. & Chilausky, R.L. (1980). Knowledge acquisition by encoding expert rules versus computer induction from examples—A case study involving soyabean pathology. International Journal of Man-Machine Studies, 12, 63-87.
- Mulhall, D.J. (1977). The representation of personal relationships: an automated system. International Journal of Man-Machine Studies, 9(3), 315-335 (May).
- Roy, J. (1986). Expert systems in Nexpert. MacTutor, 2(2), 48-51 (February).
- Shaw, M.L.G. (1980). On Becoming a Personal Scientist. London: Academic Press.
- Shaw, M.L.G. (1981). Conversational heuristics for eliciting shared understanding. Shaw, M.L.G., Ed. Recent Advances in Personal Construct Technology. pp. 31-44. London: Academic Press.
- Shaw, M.L.G. (1982). PLANET: some experience in creating an integrated system for repertory grid applications on a microcomputer. International Journal of Man-Machine Studies, 17(3), 345-360.
- Shaw, M.L.G. (1986). PCS: a knowledge-based interactive system for group problem solving. Proceedings of 1986 International Conference on Systems, Man and Cybernetics, to appear (October).

- Shaw, M.L.G. & Chang, E. (1986). A participant construct system. Proceedings of North American Personal Construct Network Second Biennial Conference. pp. 131-140. University of Calgary: Department of Computer Science (June).
- Shaw, M.L.G. & Gaines, B.R. (1983). A computer aid to knowledge engineering. Proceedings of British Computer Society Conference on Expert Systems, 263-271 (December). Cambridge.
- Shaw, M.L.G. & Gaines, B.R. (1986a). A framework for knowledge-based systems unifying expert systems and simulation. Luker, P.A. & Adelsberger, H.H., Eds. Intelligent Simulation Environments, 38-43 (January). La Jolla, California: Society for Computer Simulation.
- Shaw, M.L.G. & Gaines, B.R. (1986b). Interactive elicitation of knowledge from experts. Future Computing Systems, 1(2), to appear.
- Shaw, M.L.G. & Gaines, B.R. (1986c). An interactive knowledge elicitation technique using personal construct technology. Kidd, A., Ed. Knowledge Elicitation for Expert Systems: A Practical Handbook. To appear. Plenum Press.
- Smith, R.A. (1976). Computer-based structural analysis in the development and administration of educational materials. International Journal of Man-Machine Studies, 8(4), 439-463 (July).
- Stevens, R.F. (1985). An on-line version of the personal relations index psychological test. International Journal of Man-Machine Studies, 23(5), 563-585 (November).
- van de Brug, A, Bachant, J. & McDermott, J. (1985). Doing R1 with style. Proceedings of the Second Conference on Artificial Intelligence Applications. IEEE 85CH2215-2, pp. 244-249. Washington: IEEE Computer Society Press.
- Wahl, D. (1986). An application of declarative modeling to aircraft fault isolation and diagnosis. Luker, P.A. & Adelsberger, H.H., Eds. Intelligent Simulation Environments, pp. 25-28 (January). La Jolla, California: Society for Computer Simulation.

39-13

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

### Knowledge Base Debugging Using Apprenticeship Learning Techniques

### David C. Wilkins

Department of Computer Science Stanford University Stanford, CA 94305

## Abstract

This paper describes the ODYSSEUS apprenticeship learning program, which is designed to refine and debug any knowledge base for the HERACLES expert system shell. The input to ODYSSEUS is the observable behavior of a specialist solving a problem. ODYSSEUS analyzes the observable behavior using two underlying domain theories: a procedural theory of hypothesis-directed reasoning, and an empirical inductive theory based on past problem solving cases.

Two key components of ODYSSEUS, the global and local learning critics, are being implemented as HERACLES-based expert systems. This allows META-ODYSSEUS, which consists of copies of ODYSSEUS, to semi-automatically refine and debug ODYSSEUS in an apprenticeship learning setting. ODYSSEUS is being developed in the domain of medical diagnosis, a task domain where apprenticeship learning plays a crucial role in the creation of human experts.

# 1 Introduction

Debugging a knowledge base associated with a knowledge-intensive task is a very difficult problem, and techniques to automate this process have met with very limited success. This paper describes a solution approach for the knowledge acquisition "end-game": we assume that a reasonable knowledge base has been created, and the problem is to debug and refine this existing knowledge base.

The most effective method that human problem solvers use when confronting the "endgame" knowledge acquisition task is apprenticeship learning. For example, high-level performance in medicine is obtained by spending two or three years acquiring "first-principles" textbook knowledge and then spending five to eight years in an apprenticeship role, called an internship and residency. This is the most effective method known to create human

experts, and hence it provides motivation to investigate its efficacy in creating artificial experts.

There are many types of apprenticeship learning scenarios. One important type involves having a novice expert watch a master expert solve problems. The novice uses his or her existing knowledge to follow the line of reasoning of the expert and thereby detects when the expert is using different knowledge. If a knowledge base deficiency is suspected, the problem solving context helps the novice know what question to ask of the expert to clarify the misconception. Often the problem solving context allows the novice to identify the missing or wrong knowledge without querying the expert, though we always confirm the knowlege base change with the expert before modifying the knowlege base. Our research focuses on giving an expert system such an apprenticeship learning capability.

The ODYSSEUS learning apprentice is designed to improve any expert system that uses the HERACLES expert system shell. Our task domain is medical diagnosis, and the knowledge base we are interested in refining and debugging is the NEOMYCIN knowledge base for meningitis and neurological problems [Cla84].

# 2 The Performance Learning Model

The learning task faced by an apprentice learner will be introduced using the performance model of learning [BMSJ78] shown in Figure 1. The major functional components of the performance learning model are the performance element, the training instances generator, the intelligent editor and the learning critic. Each of these will be explained in turn.

The *performance element* to be improved is a diagnostic expert system. More specifically, the ODYSSEUS learning system is designed to improve any expert system that is implemented using the HERACLES expert system shell for classification problem solving [Cla85]. HERACLES is a generalization of NEOMYCIN [Cla84], and was created by removing the domain knowledge from NEOMYCIN.

The training instances presented to the learning system consist of the normal observable problem solving behavior of an expert. In our experiments, the expert is given a chief complaint of a patient, and the expert then makes a series of data requests to gather information that allows the patient's illness to be determined. The expert also provides clarification requests made by the learning system critic, relating to goal information. For example, the expert's goal might be to confirm a particular hypothesis or to discriminate between two likely hypotheses.

The *intelligent editor* can reason about the performance element's knowledge structures. In our case, this means it knows how to modify the HERACLES domain-specific parameter and rule files, thereby correcting bugs or adding new knowledge.



Figure 1: Performance Learning Model

The heart of the learning system is the *learning critic*. It has three functions: evaluator (global critic), diagnostician (local critic), and therapist (repair). The global critic, which is being implemented as a HERACLES-based expert system, has the responsibility of differentially modeling the specialist who is solving problems against the expert system knowledge base, and determining if an action of the specialist suggests the use of underlying knowledge that is different from that in the expert system knowledge base. For each action of the specialist, ODYSSEUS generates a set of Line of Reasoning Explanations (LOREs), and then tests them for plausibility. The most important function of the global critic is to answer a yes or no question: does an observed action of the expert suggest a knowledge base deficiency in the expert system? Since a "yes" answer relates a knowledge base deficiency to a particular action of the specialist, the global critic significantly contributes to localizing the knowledge base deficiency. The local critic, which is also being implemented as a HERACLES-based expert system, isolates where the knowledge base is buggy or incomplete and suggests a knowledge base modification. repair is straightforward: a call is made to an which knows how to modify HERACLES' knowledge

# **3** Odysseus Learning Methodology

The ODYSSEUS apprenticeship learning program has three major phases of operation. These phases are to expand the rule base and derive rule justifications, to observe actions and determine knowledge base differences, and to rationalize discrepancies and postulate new rules. This section describes the methodology of these three phases.

## 3.1 Phase 1: Expand Rule Base and Derive Rule Justifications

We assume that an initial expert system has been created manually or semi-automatically using methods such as described in [Boo84] [KNM85]. There are two ways in which this existing expert system must be augmented before differential modeling of a human specialist can commence. First, the set of heuristic rules must be expanded via induction over past problem solving cases. The original set of rules is adequate for problem solving but, in our experience, is too impoverished to model the problem solving behavior of other specialists in an apprentice context. Second, rules should be justified from first-principle knowledge or experience. Rule justifications allow a learning system to reason about the rules during the process of rationalizing discrepancies. For example, the LEAP learning apprentice for circuit design justifies rules in terms of circuit theory, a strong theory of the domain [MMS85]. By contrast, only a weak theory underlies the majority of medicine, and ODYSSEUS's justifications for heuristic rules rely on the empirical predictive power of the rules. More complex rule justifications for heuristic rules are possible, as an apprenticeship learning system for the DIPMETER ADVISOR demonstrates [SWMB85].

The induction subsystem of ODYSSEUS is principally concerned with searching the space of rules of the form  $lhs \longrightarrow hypothesis$  (CF), where CF is a MYCIN-type certainty factor. A constrained rule generator and a candidate rule evaluator find all *lhs* forms that meet given constraints of minimal rule generality (coverage), minimal rule specificity (discrimination), maximal rule colinearity (similarity), and maximal rule simplicity (number of conjunctions and disjunctions). The method of searching the space of rules is constrained rule generation from the most general hypothesis to the most specific hypothesis, and bears many similarities to META-DENDRAL [LBAL80,BM78]. The rule evaluator always gives preference to colinear forms of heuristic rules contained in the original rule base. The expanded rule set produced by the induction subsystem is incomplete; however, it bootstraps the differential modeling process that leads to its refinement. Later we will discuss how the

induction subsystem suggests missing rules to the repair subsystem during the process of rationalizing discrepancies.

## **3.2** Phase 2: Observe Actions and Detect KB Differences

Using the expanded rule base, ODYSSEUS must decide whether an action of the specialist suggests a significant domain or strategic knowledge difference between the specialist and the expert system. This is the global credit assignment problem [BM78]. The VLSI designer observed by the LEAP learning apprentice for VLSI circuit design informs LEAP when there is a missing circuit refinement rule [MMS85]. By contrast, the diagnostician's actions observed by ODYSSEUS only indirectly reflect the domain knowledge to be learned.



Figure 2: Apprenticeship learning Scenario: Apprentice is watching expert.

For each observed action of the specialist, ODYSSEUS generates a set of Line of Reasoning Explanations (LOREs). A LORE relates an an action A to an abstract strategic goal G via a skeletal rule path, that is,  $A \longrightarrow R_1 \longrightarrow R_2 \longrightarrow \cdots \longrightarrow G$ . A typical goal might be the confirmation of a particular hypothesis. Given A, all skeletal rule paths beginning with A and leading to a goal are in the LORE set of A; from the perspective of the apprentice, the set delimits the possible interpretations that can be attributed to the specialist's action A. Using the original NEOMYCIN rule base, the average is 20 and the maximum size is approximately 400.

The function of the global critic in ODYSSEUSis performed by a HERACLES-based expert system. There are three reasons why we choose to implement the global critic as an expert system. First, the task that confronts the global critic is a knowledge intensive task [DB81]. Second, with an expert system architecture, the reasoning method used by the critic can be made explicit and easily evaluated, since the knowledge is declaratively encoded using HERACLES knowledge relations. Third, since ODYSSEUS is designed to improve a HERACLES-based expert system, it can improve itself in an apprenticeship learning setting.

As we have said, ODYSSEUS generates a set of LOREs (line of reasoning explanations) for each action of the specialist. The purpose of the global critic expert system is to select one or more LOREs that correspond to the specialist's LORE or to decide that none of the LOREs provide an adequate explanation of the specialist's action. Deciding there is no adequate LORE is very difficult, since there are often weakly plausible explanations for any action of the specialist; yet to learn, the program must recognize when none of its LOREs are sufficiently plausible.

To accomplish its task, the global critic gets information that relate to LOREsfrom a variety of knowledge sources (KSs). The more important ones are: a Heracles simulator KS, multiple interpretations KS, user model KS, strategic distance KS, and a patterns of interpretation KS. A description of each of these KSs follows.

The Heracles simulator KS processes the information obtained during the problem solving session and this provides information on the current status of findings, hypotheses, and rules. Global critic rules relate these to individual LOREs. For example, if the Heracles simulator KS believes that particular hypotheses have already been confirmed or disconfirmed, then there is a rule that attaches negative evidence to all LOREs that have as a goal or subgoal the confirmation one of these hypotheses.

The multiple interpretations KS provides information that is used by global critic rules that encode heuristics used by medical domain experts to arbitrate between multiple interpretations of a specialist's action. For instance, early in the consultation session LOREs relating to more general hypotheses are preferred to LOREs with more specific hypotheses.

The user model KS records user characteristics such as individual diagnostic style preferences, and these are used by global critic rules that arbitrate between competing action justifications. For example, some problem solvers have a depth-first problem-solving style, meaning that they pursue a particular hypothesis as soon as there is weak evidence confirming it. A global critic rule would add support to those LOREs consistent with this style.

The strategic distance KS provides information that aids in determining the similarity between the expert system's preferred strategic action and the strategic action associated with each LORE. We assume that HERACLES strategic knowledge exemplifies good strategic reasoning. Therefore, if the strategic action associated with a LORE is close to HERACLES'

#### preferred strategic action, it is favored.

Finally, the *patterns of interpretation KS* provides information that is used by global critic rules that rate competing justifications according to the overall coherence they lend to the specialist's strategic plan.

If the global critic expert system is uncertain about an important fact, such as the expert's focus on a previous question in the consultation, the expert system can query the user, but this this goes against our of trying to automate the knowledge acquisition process as much as possible.

When the specialist being observed is NEOMYCIN and the rule induction phase has been omitted, ODYSSEUS always selects the correct LORE from the LORE set. However, when observing other novice and master medical specialists, if the rule induction phase is omitted, ODYSSEUS almost never selects the correct LORE; indeed, the correct LORE was not in the LORE set generated from the original NEOMYCIN knowledge base 75% of the time. This incompleteness was due to sparse domain knowledge, and motivated the initial induction phase to expand the rule base. Observing actual specialists also showed the need for more global critic rules.

### **3.3** Phase 3: Rationalize Discrepancies and Postulate New Rules

A learning opportunity exists when the ODYSSEUS global critic concludes that no LORE adequately explains the specialist's action. The task of determining the knowledge base difference is the responsibility of the local critic, which is also being implemented as a HERACLES-based expert system. The motivation for implementing the local critic as an expert system is similar to those reasons described in the previous subsection for implementing the global critic as an expert system.

The purpose of the local critic is to identify the missing or buggy knowledge relation, that will allow an adequate LORE to be constructed. In HERACLES, domain knowledge is declaratively specified by a user by instantiating knowledge relations. These knowledge relations are predicate calculus representation of domain facts. For example an instantiation of the knowledge relation template (SUGGESTS \$PARM \$HYP) is used to represent the fact that if a particular parameter is true then this suggests that a particular hypothesis is true. An instantiation of the template (ANTECEDENT-IN \$FINDING \$RULE) represents the fact that a particular finding is an antecedent in a particular rule.

The local critic selects one or more knowledge relations from among the seventy HERACLES knowledge relations, that are believed to be the reason for no LORE being found by the global critic. The information used by the local critic includes many of the KSs used by the local critic and information produced by the global critic itself. For instance, the global critic has conjectures regarding the most likely goal(s) that the expert was pursuing at

the time when the global critic concluded that no LORE explained the specialist's action. In addition, the local critic draws heavily on the induction subsystem of ODYSSEUS, when testing conjectures regarding missing or buggy knowledge. The global critic performs a bidirectional search for a skeletal path of rules between the unexplained action of the specialist and the goals conjectured by the global critic, calling on the the induction subsystem to check for rules that could connect the two search frontiers. This process aims at producing a new LORE that is considered satisfactory by the global critic.

## 4 Example

This example illustrates ODYSSEUS debugging an expert system in an apprenticeship learning situation. The domain is medical diagnosis. We are in the middle of a meningitis case which is being discussed in the presence of an apprentice. The analysis begins with question 12, at which time the apprentice believes the expert's current goal is to pursue the diagnostic hypothesis of acute bacterial meningitis. Two earlier questions will later become relevant: The expert's question 4 asked for information on headache onset; the headache onset was abrupt. The expert's question 9 asked whether the patient had meningismus, and the answer was no.

#### **Q12 of Expert:** Does the patient have a history of pneumonia?

- Q12 Odysseus Analysis: There are two potential LOREs (Line of Reasoning Explanations): testing hypotheses of diplococcus pneumoniae and mycobacterium th meningitis. The first LORE is consistent with my previous interpretation, since diplococcus pneumoniae is a subtype of acute bacterial meningitis; therefore I accept this LORE. I'll update my belief model to reflect the fact that the expert is focusing on diplococcus pneumoniae.
- Q12 Answer: Yes.
- Q13 of Expert: Does the patient have an abnormal chest X-ray?
- Q13 ODYSSEUS Analysis: There is only one potential LORE for this action: testing hypothesis diplococcus pneumoniae. This is very consistent with my previous LORE, so I am satisfied with this interpretation. I'll update my belief model to reflect the fact that the expert continues to focus on diplococcus pneumoniae.

Q13 Answer: Yes.

Q14 of Expert: Has the patient traveled recently?

Q14 ODYSSEUS Analysis: The only potential LORE is that the expert is testing hypothesis coccidioides. This is not consistent with our previous interpretation; therefore, at least one of the following is true: (a) Change of goal, (b) Missing or erroneous knowledge, (c) Previous interpretation flawed.

[Exploring (a)]. To switch from pursuing diplococcus pneumoniae to pursing coccidioides (a form of fungal meningitis) is unlikely, because of their distance apart in the disease lattice. Further, there has been no previous evidence for pursuing coccidioides. Therefore, it is unlikely that the expert's behavior is accounted for by a change in goal.

[Exploring (b)]. I asked the induction subsystem of ODYSSEUS to search for a rule sequence that links travel to diplococcus pneumoniae. None is found. I then asked the induction subsystem to determine whether a combination of previously determined findings provides motivation for pursuing coccidioides. Answer is negative.

[Exploring (c)]. Perhaps the expert was not focusing on diplococcus. But there are no LOREs other than diplococcus for question #13. There is only one alternative for question #12: mycobacterium tb meningitis. Upon being queried, the induction subsystem of ODYSSEUS confirms that abnormal chest x-ray suggests mycobacterium tb meningitis. In answer to another query, it confirms that travel to Mexico suggests mycobacterium tb meningitis. Perhaps these rules were used by the expert. If so, I will add them to my knowledge base.

The problem still is not fully resolved, because I know of no reason why the expert should be focusing on mycobacterium to meningitis. Upon request, the induction subsystem of ODYSSEUS considers all previous questions, and discovers that if headache-onset is abrupt and there is no evidence of meningismus, then this strongly suggests mycobacterium tb. I'll verify my analysis by presenting a summary to expert:

- Q14 ODYSSEUS Query: Expert, on questions 12 thru 14 were you pursuing mycobacterium tb? And did you first suspect this hypothesis based on the answers to questions 4 and 9?
- Q14 Expert Response: Well, yes. Now you're learning!

# 5 Diagnosis and Apprenticeship Learning

Figure 3 provides an interesting perspective on the classic apprenticeship learning scenario of an apprentice expert system watching an expert solve problems. In our formulation of Figure 3, there are three diagnosticians. The *first* diagnostician is the human expert, who observes the behavior of a physical system and identifies its faulty behavior. One can view the human diagnostician as differentially modeling the physical system against a mental model of the physical system. The *second* diagnostician is ODYSSEUS, which internally



Figure 3: Diagnosis everywhere. The expert is debugging a physical system, Odysseus is debugging the apprentice expert system knowledge base, and Meta-Odysseus is debugging Odysseus' (and its own) global and local critic knowledge bases.

is organized as two HERACLES-based diagnostic expert systems. ODYSSEUS differentially models the behavior of the human expert against the apprentice expert system knowledge base. When a discrepancy is discovered, ODYSSEUS assumes that the problem resides with the apprentice expert system knowledge base. The third diagnostician is META-ODYSSEUS. It comprises two expert systems, that observe and analyze the performance of the ODYSSEUS global and local critic expert systems. It is able to do this because ODYSSEUS has been fashioned to differentially model the actions of a problem solver against an expert system implemented using the HERACLES shell, and ODYSSEUS has itself been implemented as a diagnostic expert system using the HERACLES expert system shell. When ODYSSEUS makes a mistake, thus exhibiting faulty behavior, META-ODYSSEUS can attempt to isolate the problem and to correct it by modifying the ODYSSEUS' knowledge base.

# 6 The Synthetic Agent Method of Validation

We are taking an unusual approach to validating ODYSSEUS, and we hope it will lead to a general methodology for evaluating apprentice learning systems. Part of our validation approach for assessing ODYSSEUS is what one would expect: physicians will solve problems, and ODYSSEUS will watch them and attempt to recognize when they are using knowledge that is lacking from the apprentice expert system. This provides us with a lower bound on performance. We are very interested in also establishing an upper bound on performance. Our approach to this is called the *synthetic agent method*. We feel that this puts validation of an apprenticeship learning on a more principled scientific footing.

The synthetic agent method involves using an expert system with a relatively good knowledge base as a synthetic domain expert. We will refer to this expert system as the master expert system. Initially, the master and apprentice expert systems have identical knowledge bases, but then pieces of knowledge are systematically removed from the apprentice. We are interested in seeing if the apprentice can recover this knowledge or its equivalent in an apprenticeship learning situation.

We are especially interested in seeing how hard it is to learn different types of *knowledge* relations [Cla86]. As described in Section 3, domain knowledge is declaratively specified by instantiating approximately seventy different types of knowledge relations, which are a predicate calculus representation of domain facts. Clearly, an apprenticeship setting is not a panacea for human experts and we would like to quantify its limits for an apprentice expert system.

## 7 Related Research

Some of the earliest work in machine learning was directed at developing apprentice learning systems, such as Samuel's checker program [Sam63] and programs for learning LISP functions from examples [Bie76].

Davis's TEIRESIAS program addresses the knowledge acquisition problem for expert systems [Dav82]. However, the arrangement of having the specialist watch the expert system, instead of vice-versa, bypasses the machine learning problems dealing with automating the global critic, local critic, and the semantical aspects of repair.

The SEEK system for knowledge base refinement also address the problem of "end-game" knowledge acquisition [PW84]. SEEK differs from ODYSSEUS in that SEEK is a manual aid more oriented toward suggesting experiments for tuning existing knowledge than adding new knowledge. Also, SEEK does not have recourse to an underlying theory of the problem solving process.

For both SEEK and INDUCE-PLANT [Mic84], a training instance is an entire problem solving session, not a problem solving step. They are not able to recognize discrepancies based on the order in which the expert requests data during diagnosis. The training instances used by ODYSSEUS are finer-grained, and can exploit the sequence information contained in problem-solving protocols.

Waterman's exemplary programming system is directed at synthesizing an algorithmic activity by observing multiple instances of it's application [Wat78]. This contrasts to the task faced by ODYSSEUS of learning via watching heuristic problem solving. heuristic program for a domain.

The LEAP system is the premier example of an apprentice learning system for a knowledgeintensive task [MMS85]. LEAP, a learning system for VLSI circuit design, justifies rules in terms of circuit theory, a strong theory of the domain [MMS85]. By contrast, a weak theory underlies medical diagnosis, and ODYSSEUS justifies rules based on their experiential predictive power. In LEAP, the exact input and output of each problem solving step is observable; in contrast the medical specialist's actions observed by ODYSSEUS only indirectly reflects the problem-solving knowledge to be learned. The specialist solves the global critic problem for LEAP by telling the learning system whenever there is believed to be a potential knowledge discrepancy. More importantly, LEAP does not address the problem of knowledge base debugging, central to ODYSSEUS. LEAP assumes that all knowledge in the apprentice is good, that all new knowledge is guaranteed to be correct, and that no interaction of knowledge in the system can lead to problems.

## 8 Implementation Status

The implementation of the ODYSSEUS induction subsystem was completed in June 1985. The global critic's Line of Reasoning Explanation (LORE) generator is also completely implemented. An implementation of the LORE ranking subsystem exists and it correctly follows the line of reasoning of NEOMYCIN. It can also recognize situations where NEOMYCIN has used knowledge that is missing from the apprentice expert system, although this capability has not been strenuously tested yet. The global critic is presently being reimplemented as a HERACLES-based expert system: an initial vocabulary and rule set has been defined. The local critic has been designed but is not yet implemented.

# **9** Summary and Conclusions

This paper described the ODYSSEUS apprenticeship learning approach to automating "endgame" knowledge acquisition for classification expert systems. The performance learning model was used to elucidate the apprentice learning situation in Section 2, the ODYSSEUS learning methodology was described in Section 3, and 4, and an example was provided to illustrate how learning takes place in Section 5. The validation approach and related research were described in Sections 6 and 7, respectively.

Three aspects of the HERACLES expert system shell make ODYSSEUS possible: distinctions are made between the different types of knowledge in the knowledge base, such as heuristic, definitional and control knowledge [Cla83]; the method of reasoning, called hypothesisdirected reasoning, approximates that used by human experts [Cla84]; control knowledge is explicitly represented as a procedural network of subroutines and MYCIN-type metarules that are free of domain knowledge [Cla86].

ODYSSEUS has a number of distinctive features. First, because the duties of the learning critic are very knowledge-intensive, the ODYSSEUS learning critic has been formulated as two diagnostic expert systems and they are being implemented in HERACLES. Second, ODYSSEUS is designed to debug any HERACLES expert system, hence it can examine and reason about its own knowledge and knowledge structures. This ability is very helpful, because it allows ODYSSEUS to semi-automatically refine and debug itself using apprenticeship learning techniques. Third, while apprenticeship learning is a form of explanation-based learning, one of the underlying theories used heavily by ODYSSEUS is the empirical predictive power of heuristic rules. As the learning example in Section 4 demonstrates, ODYSSEUS fruitfully combines similarity based learning and explanation based learning

## 10 Acknowledgments

The advice and inspiration of Bruce Buchanan and Bill Clancey have been invaluable. Substantive comments on earlier versions of this paper by Marianne Winslett were very helpful. We are grateful for critiques of this work provided by past and present members of the GRAIL learning group at Stanford, especially Tom Dietterich and Paul Rosenbloom, and members of the GUIDON tutoring group, especially Mark Richer. Finally, we extend our thanks to physicians who have served as our domain experts: doctors Larry Fagan, Curt Kapsner, Randy Miller, Mark Musen, Roy Rada and Ted Shortliffe.

This work was supported in part by NSF grant MCS-83-12148, ONR/ARI contract N00014-79C-0302, Advanced Research Project Agency (Contract DARPA N00039-83-C-0136), the National Institute of Health (Grant NIH RR-00785-11), National Aeronautics and Space Administration (Grant NAG-5-261), and Boeing (Grant W266875). We are grateful for the computer time provided by the Intelligent Systems Lab of Xerox PARC and SUMEX-AIM.

## References

- [Bie76] A. W. Biermann. Approaches to automatic programming. In Advances in Computers, Academic Press, New York, 1976.
- [BM78] B. G. Buchanan and T. M. Mitchell. Model-directed learning of production rules. In D. A. Waterman, , and F. Hayes-Roth, editors, *Pattern-Directed Inference Systems*, pages 297-312, New York: Academic Press, 1978.
- [BMSJ78] B. G. Buchanan, T. M. Mitchell, R. G. Smith, and C. R. Johnson. Models of learning systems. In J. Belzer, editor, *Encyclopedia of Computer Science* and Technology, chapter 11, Marcel Dekker, New York, 1978. Also Stanford Report STAN-CS-79-692.
- [Boo84] J. H. Boose. Personal construct theory and the transfer of human expertise. In Proceedings of the Third National Conference on Artificial Intelligence, pages 27-33, August 1984.
- [Cla83] W. J. Clancey. The epistemology of a rule-based system: a framework for explanation. Artificial Intelligence, 20:215-251, 1983.
- [Cla84] W. J. Clancey. NEOMYCIN: reconfiguring a rule-based system with application to teaching. In W. J. Clancey and E. H. Shortliffe, editors, *Readings in Medical Artificial Intelligence*, chapter 15, pages 361–381, Addison-Wesley, Reading, Mass., 1984.
- [Cla85] W. J. Clancey. Heuristic classification. Artificial Intelligence, 27:289–350, 1985.

4(2 - 13)

- [Cla86] W. J. Clancey. Representing control knowledge as abstract tasks and metarules. In M. Coombs and L. Bolc, editors, Computer Expert Systems, Springer Verlag, 1986. Also, Knowledge Systems Lab Report KSL-85-16, Stanford University, April 1985.
- [Dav82] R. Davis. Application of meta level knowledge in the construction, maintenance and use of large knowledge bases. In R. Davis and D. B. Lenat, editors, *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill, New York, 1982.
- [DB81] T. G. Dietterich and B. G. Buchanan. The role of the critic in learning systems. Technical Report STAN-CS-81-891, Stanford University, Computer Science Dept., December 1981.
- [KNM85] G. Kahn, S. Nowlan, and J. McDermott. MORE: an intelligent knowledge acquisition tool. In *Proceedings of the Ninth IJCAI*, pages 573-580, August 1985.
- [LBAL80] R. Lindsay, B. G. Buchanan, Feigenbaum E. A., and J. Lederberg. Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project. McGraw-Hill, New York, 1980.
- [Mic84] R. S. Michalski. A theory and methodology of inductive inference. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, chapter 4, pages 83–134, Palo Alto: Tioga, 1984.
- [MMS85] T. M. Mitchell, S. Mahadevan, and L. I. Steinberg. LEAP: a learning apprentice for VLSI design. In *Proceedings of the Ninth IJCAI*, pages 573-580, August 1985.
- [PW84] P. Politakis and S. M. Weiss. Using empirical analysis to refine expert system knowledge bases. Artificial Intelligence, 22(1):23-48, 1984.
- [Sam63] A. L. Samuel. Some studies in machine learning using the game of checkers. In E. Feigenbaum and D. Feldman, editors, *Computers and Thought*, McGraw-Hill, New York, 1963.
- [SWMB85] R. G. Smith, H. A. Winston, T. M. Mitchell, and B. G. Buchanan. Representation and use of explicit justifications for knowledge base refinement. In *Proceedings of the Ninth IJCAI*, pages 673–680, August 1985.
- [Wat78] D. Waterman. Exemplary programming in RITA. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-Directed Inference Systems*, Academic Press, New York, 1978.

40+14

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

#### **Case Generation for Rule Synthesis**

Edward Wisniewski<sup>1</sup> Howard Winston Reid Smith Michael Kleyn Schlumberger-Doll Research Old Quarry Road Ridgefield, CT 06877-4108

#### ABSTRACT

We present an *Interesting Situation Generator* (ISG) that assists in the synthesis of interpretation rules from basic domain knowledge. The program discovers equivalence classes of situations (e.g., types of geological formations) that give rise to qualitatively distinct manifestations (e.g., different patterns of geophysical measurements corresponding to types of geological formations). The equivalence classes can be used by a rule generator to construct an initial set of interpretation rules of the form manifestation  $\Rightarrow$  situation.

### 1 Introduction

Traditionally, the design of knowledge-based systems has involved a process of interaction between a domain expert and a computer scientist, or knowledge engineer. During this interaction, the knowledge engineer formalizes the domain knowledge of the expert into inference rules and data structures [3]. Once the system has been implemented, failures in its performance may necessitate additional modification. As a result, the development of knowledge-based systems often involves a lengthy interaction between computer scientist and domain expert. This has been called the *knowledge acquisition bottleneck* [6]. To relieve this bottleneck, recent research in AI has focused on partially automating the knowledge acquisition process. One approach has been to develop interactive Learning Apprentice Systems (LASs). As defined in [7], such systems are 'interactive knowledge-based consultants that directly assimilate new knowledge by observing and analyzing the problem solving steps contributed by their users through their normal use of the system.'

We are currently examining the utility of the learning apprentice approach to the design of geological interpretation systems. The task for our systems is to infer the geological structures penetrated by an oil-well borehole, given a variety of petrophysical data recorded by logging tools (e.g., the tilt, or dip, of rock formations penetrated by the borehole, indexed by depth, as measured by a dipmeter tool) [10]. In [11] we discussed the utility of explicit justifications for information in the knowledge base. These justifications can be used to construct possible explanations for failures observed during routine use of the system, and help to identify faulty items to be refined in the knowledge base. We noted that these justifications contain error propagation information—how errors propagate across dependency links that relate interpretation rules to the domain theory upon which they are based.

In this paper, we address some aspects of the problem of initial knowledge base construction. Our approach

<sup>1</sup>Current address is Honeywell Systems and Research Center, 3660 Marshall Street NE, Minneapolis, MN 55418.
is to construct a set of initial *linkage rules* that embody fundamental, "textbook" knowledge of a domain [13,12]. This knowledge forms the underlying justification for a specialist's conclusions. A log interpreter, for example, may see a series of patterns and conclude that the data indicate the presence of an overturned antiform—a particular type of fold. The justification for such a conclusion can be explained by some set of basic principles, or deeper domain knowledge, combined with simplifying assumptions and approximations of the domain. Our formalization of this knowledge into *linkage rules* is one of the explicit inputs to the ISG<sup>2</sup>.

The central focus of the paper is *instance selection* [2]; that is, selection of representative problem situations to drive the rule generator during knowledge base construction and refinement. We describe an *Interesting Situation Generator* (ISG) that discovers equivalence classes of situations that give rise to qualitatively distinct manifestations.

The ISG is an active instance selector. It does not rely on an externally supplied choice of training instances. Instead, it uses the linkage rules together with error propagation information to systematically discover and operationalize [4,9] representative situations by backpropagating given manifestation classes into situation classes. Given a specific situation-manifestation pair, the largest variation in the description of the manifestation is determined such that the manifestation retains its qualitative features. By interpreting this variation as an error, the ISG can use error propagation information to determine the corresponding error in the description of the situation. By correcting the situation by this backpropagated error, it arrives at a description of a situation that lies on an equivalence class boundary. Thus, the boundaries of manifestation equivalence classes.

In the rest of this paper we will show how the ISG utilizes linkage rules that describe the way in which individual situations (e.g., such geologic structures as folds, or such pathological states in medicine as infections) produce particular manifestations of sensed data (e.g., dip of rock formations in geology, or symptoms like fever in medicine) to discover classes of situations that give rise to qualitatively similar manifestations. From these results the *rule generator* can then synthesize interpretation—or diagnostic—rules of the form manifestation  $\Rightarrow$  situation (e.g., sequence of dip patterns  $\Rightarrow$  overturned antiform fold, or fever  $\Rightarrow$  meningitis).

## 2 Related Work

Two other examples of learning systems that employed active instance selectors are Mitchell's LEX program [8] and Lenat's AM program [5]. Given a set of integral transformations, LEX could learn the heuristics that suggest the conditions under which particular transformations are useful. Its instance selector generated practice problems for the performance element to solve. The solutions to these problems were then analyzed in order to propose and refine problem solving heuristics. LEX's problem generator autonomously looked for problems that could lead to refinement of existing, but partially learned heuristics, and problems that could lead to discovery of new heuristics.

AM could discover new mathematical concepts and relations from an initial set of facts, together with a collection of heuristics for synthesizing and evaluating new facts. Its instance selector autonomously generated new tasks that either suggested new concepts to explore or new extensions to existing concepts. Task were ordered by a measure of how 'interesting' they appeared to be. Given that far more tasks could be proposed than explored, this ordering ensured that AM explored the most promising tasks first, effectively executing a best-first search through a space of mathematical concepts.

<sup>&</sup>lt;sup>2</sup>The ISG is most useful in those cases where it is easier to formalize a domain-theory as a set of linkage rules than it is to acquire a complete and consistent set of interpretation rules.

One of AM's heuristics for identifying interesting new concepts is very similar to the method our ISG uses to partition a set of situations by backpropagating the boundaries of manifestation equivalence classes. This heuristic states:

If f is a function which transforms elements of A into elements of B, and B is ordered, then consider just those members of A which are transformed into *extremal* elements of B. This set is an interesting subset of A.

If a domain theory that maps situations into manifestations is substituted for f, a set of situations is substituted for A, a set of corresponding manifestations is substituted for B, and a manifestation partition is substituted for the order relation on B, then this heuristic expresses the ISG's basic strategy for partitioning a set of situations. The interesting subset of A—discovered by the above heuristic—corresponds to the boundaries of the situation equivalence classes.

# 3 Background

We will discuss the ISG in terms of a geological domain—dipmeter interpretation. The method, however, is domain-independent and is applicable whenever (i) a domain theory is available that can map situations into their corresponding manifestations, (ii) situation and manifestation parameters belong to well-defined datatypes; and, (iii) manifestation parameters are *linear* functions of situation parameters. The last requirement can be waived if the boundaries of the set of situation classes do not have to be determined with complete accuracy. For example, this might occur in cases where errors in the initial set of interpretation rules can be recovered from during subsequent rule refinement operations.

The dipmeter records the slope, or dip, of rock layers as it ascends an oil-well borehole. These recordings can be characterized in terms of primitive patterns. Recordings of increasing dip<sup>3</sup> with borehole depth are known as red patterns, while those of decreasing dip with depth are referred to as blue patterns. The length of such a pattern is the depth at which the pattern ends minus the depth at which it begins. The rock layers of geological objects like folds and faults manifest themselves as *composites* of such patterns. One manifestation might consist of a red pattern, followed in depth by a blue pattern (a red/blue composite). Another manifestation might consist of a blue pattern followed in depth by a red pattern (a blue/red composite).

For example, Figure 1 shows a cross-section of a geologic situation, called a fold, with its associated manifestation consisting of a red/blue/red (R/B/R) composite pattern. The heavy curved line is a representative bedding layer within the fold, and the horizontal and vertical lines establish a coordinate system such that the horizontal line is parallel to the earth's surface. The entire fold is composed of layers parallel to the representative layer. If a vertical borehole penetrates this fold, it initially encounters rock layers parallel to the top flank part of the representative bedding layer. These layers all have a local slope parallel to the slope of the top tangent (tangent-1). As the borehole passes through the fold's hinge, the slope of the bedding layers it encounters first increases from the slope of tangent-1 to the slope of the vertical line (*i.e.*, 90°), then decreases from 90° to 0°, and finally increases from 0° to the slope of tangent-2. The borehole exits the fold through rock layers parallel to the bottom flank part of the representative bedding layer. These layers all have a local slope parallel to the slope of the bottom tangent (tangent-2). The top borehole interval within which the slope is *increasing* produces a *red* pattern in the corresponding dipmeter log, the middle interval within which the slope is *decreasing* produces a *blue* pattern, and the bottom interval within which the slope is *increasing* produces another *red* pattern. This explanation assumes continuity of the subsurface region. Also, the angle at which the borehole penetrates the region will affect the dipmeter recordings. Nevertheless,

41-Z

<sup>&</sup>lt;sup>3</sup>Patterns of monotonically increasing, decreasing, or constant dip are required to have uniform azimuth. That is, the dip direction must remain constant.



Figure 1: Change in Dip as a Function of Depth

this is the type of explanation that might justify why a geological structure manifests itself by a particular pattern.

Figure 2 shows an example of a linkage rule that maps geologic regions having a certain type of geometric structure into their corresponding R/B dipmeter patterns. All these regions have their top-tangent (*i.e.*, tangent-1) in quadrant III and their bottom-tangent (*i.e.*, tangent-2) in quadrant IV. The slope of a line swept from the top-tangent to the bottom-tangent first increases (red-pattern), then decreases (bluepattern), and finally increases again (red-pattern). The top dip of a red or blue pattern is defined as the dip magnitude at the depth at which the pattern starts, and the bottom dip of a red or blue pattern is defined as the dip magnitude at the depth at which the pattern ends. Clause A-3 sets the top-dip of the red-pattern to be the angle between tangent-1 and -x axis, and clause A-6 sets the bottom-dip of the blue-pattern to be the angle between tangent-2 and the +x axis. Clause A-4 assigns the bottom-dip of the red-pattern to be 90°, and clause A-5 assigns the top-dip of the blue-pattern to be 90°.

A set of manifestations can be partitioned into a collection of equivalence classes by specifying an equivalence relation over the manifestations. For example, let two manifestations be equivalent iff they consist of the same sequence of primitive patterns. Then one equivalence class would be the set of all manifestations that consist of red/blue composites. In this equivalence class, the length of the primitive patterns is not a determining factor for membership, because the equivalence relation does not require the lengths of corresponding primitive patterns within two red/blue composites to be equal in order for the two composites to be equivalent. For example, manifestations consisting of red/blue composites with short red patterns are equivalent, or qualitatively similar, to manifestations consisting of red/blue composites with long red patterns.

Associated with any situation or manifestation is a set of parameters. Situation parameters are those properties of the most specific situation classes within the domain theory that are left unspecified. A fold, for example, can be described by its fold angle  $\theta_f$  (the angle between the flanks of the fold) and its orientation  $\theta_o$  with respect to the earth's surface (see Figure 1). If any of these properties are left unspecified in the

IF .	(C-1 C-2 C-3)						
C-1	(There-exists a Region :R)						
C-2	(Tangent-1 of :R lies in Quadrant III)						
C-3	(Tangent-2 of :R lies in Quadrant IV))						
THEN	(A-1 A-2 A-3 A-4 A-5 A-6)						
A-1	(Create a Red-Pattern :Red)						
A-2	(Create a Blue-Pattern :Blue)						
A-3	(Assign the Top-Dip of :Red to be (- (The Orientation of Tangent-1)						
	180))						
A-4	(Assign the Bottom-Dip of :Red to be 90)						
A-5	(Assign the Top-Dip of :Blue to be 90)						
A-6	Assign the Bottom-Dip of :Blue to be (- 360						

(The Orientation of Tangent-2)))

#### Figure 2: R/B Linkage Rule

description of fold structures, then they are parameters of fold situations. For example, if folds in some domain model were only differentiated according to their fold angle, then orientation would be the only parameter of fold situations.

Manifestation parameters are defined with respect to some partition of the set of manifestations. They are manifestation properties that can be used to distinguish, or index, members of the partition's equivalence classes. For the manifestation partition described above, we chose primitive pattern lengths to be the manifestation parameters. For example, in this case, a red/blue composite is a manifestation whose parameters are the lengths of its component red and blue patterns. Note that we could have alternatively chosen the top and bottom dips of the primitive patterns to be the manifestation parameters because they can also be used to distinguish manifestations within the same equivalence class. In the worked example, presented in Section 4, top and bottom dips were used instead of pattern lengths because dip values have lower  $(0^{\circ})$  and upper bounds  $(90^{\circ})$  whereas pattern lengths only have lower bounds (0). As explained below, this provides more constraints on determining the boundaries of situation equivalence classes.

The ISG algorithm requires that the *datatype* of each situation and manifestation parameter be an ordered set of values for which a metric is defined. That is, the domains of situation parameters and the ranges of manifestation parameters must be partially ordered sets (posets), and the distance between any two values of a parameter must be defined. In our example, the domain of fold angles  $\theta_f$  is the interval (0°, 180°), the domain of fold orientations  $\theta_o$  is the interval (0°, 90°), and the domain of primitive red and blue pattern lengths is the interval (0,  $\infty$ ). The order relation on these intervals is the standard ordering of the real numbers (*i.e.*,  $a \leq b$  iff b - a is nonnegative), and the metric is the usual metric for the real numbers (*i.e.*, d(a, b) = |b - a|). As explained below, the ISG uses this information to (*i*) define maximal parameter errors; and, (*ii*) check the validity of parameter values.

For the ISG to be applicable, the domain theory must define *causal relations* between situation and manifestation parameters. That is, it must be possible to infer the values of manifestation parameters based on the values of situation parameters. Suppose some critical value of a situation parameter causes a manifestation parameter to assume a value that represents the lower or upper bound of its range. By definition, the manifestation having this extreme value for one of its parameters lies on the boundary of a manifestation equivalence class. Thus, situations having parameter values less than the critical value must give rise to a different set of manifestations than situations with parameter values greater than the critical value.

The ISG uses causal relations between situation and manifestation parameters to differentiate situation classes with respect to critical values of their situation parameters. Each differentiated subclass corresponds



Figure 3: Differentiated Folds



Table 1: The ISG Problem

to a single manifestation. Thus, the ISG induces a partition of situation subclasses from a given partition of manifestations. All the possible manifestations of the given situation class are determined and each is associated with one or more of the situation's induced subclasses.

# 4 The ISG Algorithm

## 4.1 Input/Output Behavior

This section will explain how the ISG works by showing how it generates the tree of fold classes in Figure 3. As shown in the hierarchy, Folds are classified as Acute, Right, or Obtuse depending on the value of their fold angle  $\theta_f$ . Each of these subclasses can in turn be divided according to the fold's orientation  $\theta_o$ . Each of the terminal situation classes shown in Figure 3 has a single manifestation m. For example, the manifestation of class Acute-1 is a composite R/B pattern. The input/output behavior of the ISG is summarized in Table 1.

Given a domain theory of linkage rules L that map situations in S into manifestations in M, a set of



Figure 4: Fold Parameter Space

manifestation parameters, and an ordering of S's situation parameters, the ISG finds the partition  $\pi_S$  of S induced by a given partition  $\pi_M$  of M.

In our example, the situation class S is the set of folds parametrized by fold angle  $\theta_f$  and orientation  $\theta_o$  (*i.e.*, the Fold class shown in Figure 3). The fold angle must lie between 0° and 180°, and the fold orientation must lie between 0° and 90°. We will order these parameters such that the situation partition  $\pi_S$  produced by the ISG first differentiates S with respect to  $\theta_o$  and then with respect to  $\theta_f^4$ . The manifestation class M is the set of primitive red and blue patterns together with composite patterns made up of sequences of red and blue patterns. The set of linkage rules L is a simple geologic model that maps folds in S into their corresponding manifestations in M. The set of manifestations is given a partition  $\pi_M$ , such that any two manifestations are equivalent iff they are composed of the same sequence of red and blue patterns. In this example, the top and bottom dips of the red and blue patterns comprising a manifestation will be used as the manifestation parameters. Given the above information, the ISG will find the partition  $\pi_S$  of S shown in Figure 3.

In this example, the ISG will explore the situation parameter space shown in Figure 4. Regions of the parameter space are labelled with the manifestations they produce, and the situations corresponding to some representative points are illustrated in Figure 5; where only the orientations of the top and bottom tangents with respect to the horizontal and vertical axes are shown. For example, Figure 5 (g) is a representation of the situation depicted in Figure 1.

In the following, for any manifestation M, the set of manifestations that are members of the same equivalence class as M will be denoted [M] and will be called the equivalence class generated by M. Similarly, for any situation S, the set of situations that are members of the same equivalence class as S will be denoted [S] and will be called the equivalence class generated by S. The boundary of [S] (*i.e.*, those elements of [S] with the

<sup>&</sup>lt;sup>4</sup>If the order of these parameters were reversed, the ISG would grow a tree of fold classes such that the top-level differentiated folds with respect to  $\theta_j$ . The particular ordering selected for our example was chosen so that the situation equivalence classes discovered by the ISG correspond to those typically used by geologists [1].



Figure 5: Fold Situations

I	fold	l	8-13-1	1	fold-1-1	
	$\theta_{f}:(0.180)$		θ+:45		09:40	
	f.:(0.90)		fa:(0.90)		θ <sub>o</sub> :10 m:R/B	
	• • • • • • • • • • • • • • • • • • • •					ł

Figure 6: ISG Intermediate Results - 1

largest or smallest values of the situation parameters) will be denoted as  $\partial[S]$ . Similarly, the boundary of [M] (*i.e.*, those elements of [M] having the largest or smallest values of the manifestation parameters) will be denoted as  $\partial[M]$ . The goal of the ISG is to determine  $\partial[S]$  from S, M, and  $\partial[M]$ .

#### 4.2 Getting Started

The ISG starts by randomly selecting an initial situation  $S_0$  from the Fold class. Suppose this situation corresponds to point  $p = (\theta_o^0, \theta_f^0) = (10^\circ, 45^\circ)$  in the situation parameter space. By applying the linkage rules L to  $S_0$ , the ISG determines that this situation produces a particular R/B manifestation  $M_0$ .

With this information, the ISG begins to grow (depth-first) the tree in Figure 3 by creating the structure shown in Figure 6. Node fold-1 represents what will become the Acute class and node fold-1-1 represents what will become the Acute-1 subclass. For now, node fold-1-1 represents the initial situation  $S_0$ .

## 4.3 Error Propagation Information and the Inverse–Problem for Rule–Based Systems

The ISG uses error propagation information to find situation equivalence classes that correspond to given manifestation equivalence classes. Given an initial situation—manifestation pair  $(S_0, M_0)$ , the ISG determines the situation parameter errors that correspond to the largest errors that can be made in the manifestation parameters, such that the qualitative features of  $M_0$  are preserved. By correcting  $S_0$  by these backpropagated situation parameter errors, the ISG finds the situations that define  $\partial [S_0]^5$ . This section briefly describes the nature of error propagation information and its use in addressing the inverse-problem for rule-based systems.

When a rule is successfully applied to a knowledge base, information is generated about the way in which potential errors in the rule and knowledge base could be propagated to the rule's conclusions. To take a simple example, suppose we apply a rule to calculate the area of a rectangle from its height and width. Obviously, over-estimating or under-estimating the height or width of the rectangle will produce an overestimation or under-estimation, respectively, in its area. Such error propagation information is stored in the data structure that records the support, or justification, of the the rule's conclusion. If the application of other rules depends on the area of the rectangle, then the error in its value may propagate to the conclusions made by these rules. Thus, during the course of rule application, the error propagation generator determines the way in which errors propagate from one rule application to another.

Error propagation information is locally determined with respect to the values referenced and created by

<sup>&</sup>lt;sup>5</sup>As explained in Section 4.4, the ISG also forward propagates maximum situation parameter errors in an effort to extend the boundaries of  $[S_0]$  that are discovered by backpropagation.

a rule application. That is, if a rule computes a value of  $y = y_o$  based on values of  $x_i = x_{i,o}$ , then the associated error propagation information is essentially  $(\partial y/\partial x_i)$  evaluated at  $x_i = x_{i,o}$ , and tells us how deviations of  $x_i$  from  $x_{i,o}$  would propagate to deviations in y from  $y_o$ ; provided the other justifiers of y (i.e.,  $x_{j\neq i}$ ) are held constant. For example, in the rule for computing the area of a rectangle as  $A = H \times W$ , the error propagation information relating H and A states that an error  $\delta H$  in H would cause an error  $\delta A$  in A such that  $\operatorname{sign}[\delta A] = \operatorname{sign}[(\partial A/\partial H)\delta H] = \operatorname{sign}[W] \operatorname{sign}[\delta H] = \operatorname{sign}[\delta H]$ , and  $|\delta A| = |(\partial A/\partial H)\delta H| = W$   $|\delta H|$ . Thus, the induced error in A depends upon the value of W that obtained when the rule was applied.

The ISG is a method for finding the inverse-image of parameter values that characterize manifestations on the boundary of equivalence classes. Because the linkage rules connecting situations to manifestations cannot in general be inverted, the ISG uses error propagation information to backpropagate these values by backpropagating their differences from reference values<sup>6</sup>. If these differences are interpreted as errors in the reference values, error propagation information can reveal the corresponding error between the unknown situation parameter values and the reference values that correspond to the reference manifestation parameter values<sup>7</sup>. By correcting the reference situation parameter values by this backpropagated error, the unknown parameter values characterizing boundary situations can be determined. As explained in Section 5 below, the locality of error propagation information restricts the accuracy with which the unknown situation parameter values can be calculated.

#### 4.4 Finding the First Situation Equivalence Class

Given  $S_0$  and  $M_0$ , the ISG will backpropagate the largest possible errors in  $M_0$  and forward propagate the largest possible errors in  $S_0$  to determine  $\partial[S_0]$ . The maximum manifestation errors are the largest changes that can be made to  $M_0$ 's parameter values such that the resulting manifestation is still qualitatively similar to  $M_0$  (*i.e.*, these changes correspond to the distance between  $M_0$ 's parameter values and the parameter values of the manifestations that lie on  $\partial[M_0]$ ). Maximum situation errors are the largest changes that can be made to  $S_0$ 's parameter values such that the resulting situation is not an element of a previously determined equivalence class.

Error propagation information (generated when  $S_0$  was mapped into  $M_0$ ) records the way in which changes in the situation parameters are related to changes in the manifestation parameters. This information will be used to find the interval of orientations surrounding  $\theta_o^0 = 10^\circ$  such that any situation having an orientation within this interval and fold angle  $\theta_f^0 = 45^\circ$  will produce a manifestation qualitatively similar to  $M_0$  (*i.e.*, a R/B manifestation that is a member of  $[M_0]$ ). Figure 7 shows how this is accomplished. The situation parameters are  $\theta_o$  and  $\theta_f$ . The manifestation parameters,  $R_{top}$ ,  $R_{bot}$ ,  $B_{top}$ , and  $B_{bot}$ , are the top and bottom dips of the red and blue patterns that comprise the R/B manifestation. Error propagation information relates the way in which changes in  $\theta_o$  are related to changes in these manifestation parameters. This information will be used to find the values of  $\theta_o$  that correspond to the maximum (90°) and minimum (0°) values of the manifestation parameters. Any of these backpropagated  $\theta_o$  values that lie within previously explored regions of its domain will be ignored<sup>8</sup> The interval around  $\theta_o$  that determines  $[S_0]$  will be obtained by selecting the *smallest* backpropagated value greater than  $\theta_o$  and the largest backpropagated value smaller than  $\theta_o$ . The largest and smallest values of  $\theta_o$ , not already included in an equivalence class interval, will be added to this set of backpropagated values if they forward propagate into valid manifestation parameter values.

In Figure 7, the manifestation parameter values corresponding to  $(\theta_f^0, \theta_o^0) = (45^\circ, 10^\circ)$  are indicated by the

<sup>&</sup>lt;sup>6</sup>The manifestation parameter values characterize manifestations on  $\partial[M_0]$ , and the reference manifestation parameter values characterize  $M_0$ .

<sup>&</sup>lt;sup>7</sup>The unknown situation parameter values characterize situations on  $\partial[S_0]$ , and the reference situation parameter values characterize  $S_0$ .

<sup>&</sup>lt;sup>8</sup>This occurs for those  $\theta_o$  values obtained from backpropagating manifestation parameter extrema that do not characterize  $\partial[S_0]$ .



Figure 7: Extending Fold-1-1

solid circles (•). The arrows above these points show how decreasing  $(\leftarrow)$  or increasing  $(\Rightarrow)$   $\theta_o$  affects these values. That is, they illustrate how errors in situation parameter values propagate to errors in manifestation parameter values. The manifestation parameters change in the direction indicated by the single-arrows (double-arrows) when  $\theta_o$  is decreased (increased). For example, *increasing*  $\theta_o$  from  $\theta_o^0$  causes  $R_{top}$  to *increase* from  $\beta$  and  $B_{bot}$  to decrease from  $\delta$ . Note that  $R_{bot}$  and  $B_{top}$  are not affected by altering  $\theta_o$ . As explained below, the open circles ( $\circ$ ) show the results of forward propagating extreme situation parameter errors and backpropagating extreme manifestation parameter errors.

The equivalence class  $[S_0]$  will be determined in two steps.

- 1. Elements of  $\partial[M_0]$  will be backpropagated, using error propagation information, to find elements of  $\partial[S_0]$ . By definition, manifestations on  $\partial[M_0]$  are manifestations one of whose parameters has either the largest or smallest value of all manifestations in  $[M_0]$ . Situations on  $\partial[S_0]$  are defined to be those situations in  $[S_0]$  that correspond to manifestations on  $\partial[M_0]$ .
- 2. The ISG tries to extend  $\partial[S_0]$  (obtained by backpropagating extreme  $M_0$  parameter errors in step 1) by forward propagating extreme  $S_0$  parameter errors. This is accomplished by checking to see if the largest or smallest values a situation parameter can assume (*i.e.*, the endpoints of the largest interval surrounding a situation parameter value that doesn't include values characterizing other equivalence classes) map into valid values of all the manifestation parameters. If so, the extreme situation parameter value is taken as belonging to a situation on  $\partial[S_0]$ .

#### Step 1—Backpropagation

- Because  $R_{bot}$  and  $B_{top}$  are equal to 90° for all R/B patterns, they are not effective in distinguishing members of  $[M_0]$ , so that forward and backward propagation of maximal errors does not take place between  $\theta_o$  and these manifestation parameters. Thus, these manifestation parameters do not play a role in determining  $[S_0]$ .
- However, the elements in  $[M_0]$  can be distinguished by their  $R_{top}$  and  $B_{bot}$  values. By definition, the elements of  $[M_0]$  that lie on the  $\partial[M_0]$  have the largest or smallest values of these parameters. If  $R_{top} = 0^\circ$  characterizes one of these boundary points, then it will backpropagate into a valid value of  $\theta_o$  that characterizes a situation on  $\partial[S_0]$ . That is, if  $R_{top} = \beta$  suffers the largest possible parameter overestimation error such that it should equal 0°, then the corresponding error in  $\theta_o$  should put its corrected value between 0° and 90°. However, using error propagation information, we find that correcting  $\theta_o$  would cause it to be less than 0° and lie outside the domain of valid fold orientations. Thus, a manifestation with  $R_{top} = 0^\circ$  does not lie on  $\partial[M_0]$  and cannot be used to determine a  $\theta_o$  value of a situation that lies on  $\partial[S_0]$ . Similarly, if  $B_{bot} = 90^\circ$  for an element of  $\partial[M_0]$ , then  $\theta_o$  would also be less than 0° the corresponding element of  $\partial[S_0]$ . Because this lies outside the domain of  $\theta_o$ , there is no manifestation with  $B_{bot} = 90^\circ$  on  $\partial[M_0]$ , and this point is discarded in looking for  $\partial[S_0]$ .
- However,  $R_{top} = 90^{\circ}$  backpropagates into  $\theta_o = a'$ , and  $B_{bot} = 0^{\circ}$  backpropagates into  $\theta_o = a$ ; where  $\theta_o^0 < a < a'$ . The smallest of these two backpropagated values greater than  $\theta_o^0$  is selected as the upper bound of  $\theta_o$  values for situations in  $[S_0]$ . This is because values of  $\theta_o$  between a and a' would map into valid values of  $R_{top}$  between some value  $\gamma$  and 90°, but would also map into invalid values of  $B_{bot}$  less than 0° (see Figure 7).

The ISG has at this point discovered the upper bound of the interval surrounding  $\theta_o^0$  that contains values of  $\theta_o$  belonging to elements of  $[S_0]$ . This upper bound is  $\theta_o = a$ , and characterizes a situation on  $\partial [S_0]$ . Although backpropagation did not extend the interval below  $\theta_o^0$ , the next step will.

4(-11

#### Step 2—Forward Propagation



Figure 8: Situations in  $[S_0]$ 

- The smallest value of  $\theta_o$  less than 10° that is not already included in another situation equivalence class is 0°. Using error propagation information, the ISG determines that this value maps into  $R_{top} = \alpha$ , and  $B_{bot} = \alpha$ . These are valid values for manifestation parameters. Therefore, the interval (10°, a) is extended to (0°, a). We now know that  $[S_0]$  situations with  $\theta_o$  in (0°, a) and  $\theta_f = 45^\circ$  map into the same equivalence class of R/B patterns. Note that, in general, a will depend on  $\theta_f$ .
- The largest value of  $\theta_o$  greater than 10° that is not already included in another situation equivalence class is 90°. We do not have to forward propagate this value because we have already extended the interval to the right of  $\theta_o^0$  by a smaller amount. That is, we already determined that a situation with  $90^\circ > \theta_o = a > \theta_o^0$  lies on  $\partial [S_0]$ .

The above procedure for determining  $[S_0]$  can be given a geometric interpretation that makes the application of the ISG to this example clearer. The initial situation  $S_0$  corresponds to point p in Figure 4, and its corresponding situation is represented by diagram p in Figure 8.

Manifestation parameter  $R_{top}$  is the dip of tangent-1 (equal to  $\beta$  in  $S_0$ ),  $R_{bot}$  and  $B_{top}$  are both the dip of the vertical line (90°), and  $B_{bot}$  is the dip of tangent-2 (equal to  $\delta$  in  $S_0$ ). For situation  $S_0$  at point p,  $R_{top}$  is greater than  $B_{bot}$  as shown by  $R_{top} = \beta > \delta = B_{bot}$  in Figure 7.

The following equations relate the geometric and situation parameters.

$$\alpha = (1/2)\theta_f$$
  

$$\beta = (1/2)\theta_f + \theta_o$$
  

$$\delta = (1/2)\theta_f - \theta_o$$
  

$$\gamma = \theta_f$$

Suppose we start with situation  $S_0$  shown in diagram p in Figure 8.

• If the orientation  $\theta_o$  is increased by rotating the tangent vectors counter-clockwise, the dip of tangent-1 increases and the dip of tangent-2 decreases. When  $\theta_o$  reaches some value a, the dip of tangent-2  $(B_{bot})$ becomes 0°. This corresponds to the situation depicted in diagram q in Figure 8. The dip of tangent-1  $(R_{top})$  becomes equal to  $\gamma$ , and the bottom dip of the red pattern and the top dip of the blue pattern remain equal to 90°. • Starting with  $S_0$ , if the orientation  $\theta_o$  is decreased by rotating the tangent vectors clockwise, the dip of tangent-1 decreases and the dip of tangent-2 increases. When  $\theta_o$  becomes equal to 0°, the dips of tangent-1 and tangent-2 become equal to  $\alpha$  (i.e.,  $R_{top} = B_{bot} = \alpha$ ). Again, the bottom dip of the red pattern and the top dip of the blue pattern remain equal to 90°. This corresponds to the situation depicted in diagram f in Figure 8.

Using Figure 8, we can redo step 1 of the ISG algorithm by reinvestigating the situations corresponding to extrema of the R/B manifestation equivalence class  $[M_0]$ .

- $R_{top} = 0^{\circ}$  If  $R_{top}$  equals 0°, tangent-1 would be horizontal. To obtain this situation, the tangents in diagram f would have to be further rotated clockwise beyond  $\theta_o = 0^{\circ}$ . Thus, the orientation  $\theta_o$  would be negative. Because this lies outside the domain of  $\theta_o$ , a manifestation with  $R_{top} = 0^{\circ}$  is not on  $\partial [M_0]$ .
- $B_{bot} = 90^{\circ}$  Similarly, if  $B_{bot}$  equals 90°, tangent-2 would be vertical, and the orientation would again be negative. Thus, a manifestation with  $B_{bot}$  equal to 90° cannot lie on  $\partial[M_0]$ .
- $B_{bot} = 0^{\circ}$  If  $B_{bot}$  equals 0°, tangent-2 would be horizontal. The corresponding manifestation must lie on  $\partial[M_0]$ , because if the orientation is further increased a qualitatively new R/B/R manifestation arises (see Figure 5 diagram g). Thus, the value of  $\theta_o$  that corresponds to  $B_{bot} = 0^{\circ}$  gives the upper bound of  $\theta_o$  values in  $[S_0]$ , and hence a situation on  $\partial[S_0]$ .
- $R_{top} = 90^{\circ}$  If  $R_{top} = 90^{\circ}$ , tangent-1 would be vertical. To obtain this situation, the tangents in diagram f would have to be rotated counter-clockwise such that  $\theta_o > a$ . However, as explained above, manifestations of such situations are R/B/R patterns that lie outside of  $[M_0]$ . The ISG excludes these situations by taking the smallest backpropagated value of  $\theta_o$  greater than  $\theta_o^{\circ}$  (*i.e.*, taking a as the value of  $\theta_o$  for a situation on  $\partial[S_0]$ ).

Again, using Figure 8, we can redo step 2 of the ISG algorithm by reinvestigating the manifestations that correspond to extrema of  $S_0$ 's parameters.

- $\theta_o = 0^\circ$  If the orientation of  $S_0$  is decreased from 10° to 0°, we obtain the situation shown in diagram f in Figure 8. Its corresponding manifestation is still a R/B pattern that belongs to  $[M_0]$ , and we make situation f the lower bound of  $[S_0]$ .
- $\theta_o = 90^\circ$  If the orientation of  $S_0$  is increased from 10° to 90°, we obtain situation h in Figures 4 and 5. The corresponding manifestation is now a B/R pattern that lies outside of  $[M_0]$ . Thus, a situation with  $\theta_o = 90^\circ$ , must lie outside of  $[S_0]$ .

### 4.5 Finding other Progeny of Fold-1

At this point, we can extend the tree of partial results shown in Figure 6 into that shown in Figure 9 by extending the orientation of the fold-1-1 class from  $\theta_o = 10^\circ$  to  $\theta_o \in (0^\circ, a)$ .

To continue partitioning the domain of  $\theta_o$ , a new child of the fold-1 class (*i.e.*, fold-1-2) is created having a  $\theta_o$  value that lies in the unexplored region outside of  $(0^\circ, \mathbf{a})$  For example, this new situation  $S_1$  might correspond to  $(\theta_o^1, \theta_f^1) = (40^\circ, 45^\circ)$ ; where  $40^\circ > a$ . If the method by which situation  $S_0$  was extended to  $[S_0]$  is applied to situation  $S_1$ , the ISG will find that members of  $[S_1]$  have values of  $\theta_o$  in the interval  $(\mathbf{a}, \mathbf{b})$ . Here, b (like a) depends upon  $\theta_f$ . Because the fold-1-1 and fold-1-2 subclasses of fold-1 do not yet cover the entire domain of fold-1 orientations, a third child of the fold-1 class (*i.e.*, fold-1-3) is created that represents a situation  $S_2$  having a  $\theta_o$  value in the interval  $(\mathbf{b}, 90^\circ)$ . Node fold-1-3 is expanded into  $[S_2]$ . It turns out that values of  $\theta_o$  for members of this equivalence class lie in the interval  $(\mathbf{b}, 90^\circ)$ . At this

			_	fold-1-1	
fold		fold-1		θ <sub>f</sub> :45	
$\theta_f:(0,180)$	<u> </u>	θ <sub>f</sub> :45		θo:(0,a)	
.θο:(0,90)		θo:(0,90)		m:R/B	}

Figure 9: ISG Intermediate Results - 2



Figure 10: ISG Intermediate Results - 3

point, the ISG has found a complete set of fold-1 subclasses such that each subclass corresponds to a single manifestation (see Figure 10).

#### 4.6 Extending the Fold-1 Class

The ISG now tries to extend the fold-1 class by looking for an interval surrounding  $\theta_f^0 = 45^\circ$  such that as  $\theta_o$  is varied from 0° to 90° the same sequence of manifestations (R/B, R/B/R, B/R) are produced. That is, the ISG tries to enlarge fold-1 to include situations with fold angles other than 45° that have the same set of subclasses fold-1-1, fold-1-2, and fold-1-3. Figure 11 shows how this is done.

Here,  $\theta_o(R/B)$  represents the proportion of  $\theta_o$  values that correspond to R/B manifestations (*i.e.*, a/90),  $\theta_o(R/B/R)$  represents the proportion of  $\theta_o$  values that correspond to R/B/R manifestations (*i.e.*, (b-a)/90), and  $\theta_o(B/R)$  represents the proportion of  $\theta_o$  values that correspond to B/R manifestations (*i.e.*, (90-b)/90). These are second-order manifestation parameters because they distinguish, or index, members of secondorder manifestation equivalence classes<sup>9</sup>. First-order manifestation equivalence classes were composed of sequences of primitive red and blue patterns. For example, the set of sequences of red followed by blue patterns comprise the R/B first-order manifestation equivalence classes. For example, the set of sequences of R/B, followed by R/B/R, followed by B/R manifestations comprise the second-order (R/B, R/B/R, B/R) manifestation equivalence class. Each element of this class can be distinguished by the

<sup>&</sup>lt;sup>9</sup>The ISG actually uses the unnormalized lengths of  $\theta_o$  intervals occupied by first-order manifestations as its set of second-order manifestation parameters. In this case, second-order manifestation parameters would range from 0° to 90°.



Figure 11: Extending Fold-1

fractions of the  $\theta_o$  interval (0°, 90°) that are occupied by the R/B, R/B/R, and B/R patterns; where

 $\theta_o(R/B) + \theta_o(R/B/R) + \theta_o(B/R) = 1.$ 

In Figure 11, the fraction of  $\theta_o$ 's range that corresponds to a R/B, R/B/R, or B/R manifestation for  $\theta_f^0 = 45^\circ$ are indicated by the solid circles (•), and the arrows above these points show how decreasing ( $\leftarrow$ ) or increasing ( $\Rightarrow$ )  $\theta_f$  affects these fractions. For example, *increasing*  $\theta_f$  from  $\theta_f^0$  causes  $\theta_o(R/B)$  to *increase*,  $\theta_o(R/B/R)$ to *decrease*, and  $\theta_o(B/R)$  to *increase*. As explained below, the open circles (•) show the results of forward propagating extreme situation parameter errors and backpropagating extreme second-order manifestation parameter errors.

The fold-1 class is extended by applying the ISG algorithm to the situation parameter  $\theta_f$  and the secondorder manifestation parameters  $\theta_o(R/B)$ ,  $\theta_o(R/B/R)$ , and  $\theta_o(B/R)$ . Elements on  $\partial[(R/B, R/B/R, B/R)]$ have the largest or smallest values of the second-order manifestation parameters. Backpropagating these extreme values and forward propagating extreme situation parameter values give us the values of  $\theta_f$  that characterize situations on  $\partial[\text{fold-1}]$ .

In order to use the ISG algorithm to extend the fold-1 class, second-order error propagation information is needed that relates changes in  $\theta_f$  to changes in  $\theta_o(R/B)$ ,  $\theta_o(R/B/R)$ , and  $\theta_o(B/R)$ . This information is empirically determined by recomputing the boundaries of the R/B, R/B/R, and B/R equivalence classes as  $\theta_f$  is increased and decreased from  $\theta_f^0 = 45^\circ$ . For example, the upper bound of  $\theta_o$  values (i.e., a) that characterizes the situation whose corresponding manifestation lies on  $\partial[R/B]$  depends upon the situation's fold angle  $\theta_f$ . Values of this upper bound are recomputed for  $\theta_f = \theta_f^0 + 1$  and for  $\theta_f = \theta_f^0 - 1$ . Because the upper bound (a) increases and the lower bound (0°) remains constant as  $\theta_f$  increases,  $\theta_o(R/B)$  increases as  $\theta_f$ increases. Thus, a parameter over-estimation or under-estimation error in  $\theta_f$  would cause a parameter overestimation or under-estimation error, respectively, in  $\theta_o(R/B)$ . The same procedure is followed to determine how errors propagate between  $\theta_f$  and the other second-order manifestation parameters  $\theta_o(R/B/R)$  and  $\theta_o(B/R)$ .



Figure 12: ISG Intermediate Results - 4

#### Step 1—Backpropagation

Extreme values of the second-order manifestation parameters (i.e., 0 and 1) are backpropagated into the domain of  $\theta_f$ . In this case, if  $\theta_o(R/B) = 0$ ,  $\theta_o(R/B/R) = 1$ , or  $\theta_o(B/R) = 0$ , then  $\theta_f = 0^\circ$ . If  $\theta_o(R/B) = 1$ ,  $\theta_o(R/B/R) = 0$ , or  $\theta_o(B/R) = 1$ , then  $\theta_f = 90^\circ$ . These backpropagated values are combined by

- taking the smallest backpropagated value of  $\theta_f$  greater than  $\theta_f^0 = 45^\circ$  (i.e., 90°) as the fold angle of an element on  $\partial$ [fold-1], and
- taking the largest backpropagated value of  $\theta_f$  smaller than  $\theta_f^0 = 45^\circ$  (i.e., 0°) as the fold angle of an element on  $\partial$ [fold-1].

Thus, [fold-1] now consists of those situations having a fold angle  $\theta_f$  between 0° and 90° (*i.e.*, the Acute class in Figure 3).

#### Step 2—Forward Propagation

- The largest value of  $\theta_f$  greater than 45° that is not already included in another situation equivalence class is  $\theta_f = 180^\circ$ . If this value is forward propagated, it will map into invalid values of the second-order manifestation parameters. Thus, the upper bound of the interval of  $\theta_f$  values determined by step 1 (90°) is not extended to 180°.
- The smallest value of  $\theta_f$  smaller than 45° that is not already included in another situation equivalence class is  $\theta_f = 0^\circ$ . Because this value is already included in the interval of  $\theta_f$  values that characterize situations in [fold-1], it is not forward propagated into the range of second-order manifestation parameters.

After extending the fold-1 class, the ISG's intermediate results are shown in Figure 12. The interval of  $\theta_f$  values (0°, 90°) characterize a situation equivalence class that produces the same sequence of first-order manifestations (R/B, R/B/R, B/R) as  $\theta_o$  is varied through its domain from 0° to 90°. In other words, we have distinguished Acute folds as the set of situations whose manifestations go through the sequence R/B  $\rightarrow$  R/B/R  $\rightarrow$  B/R as their orientation is increased from 0° to 90°.

### 4.7 Completing the Example

At this point, the ISG has grown the branch of Acute folds in Figure 3; where fold-1 corresponds to Acute, fold-1-1 corresponds to Acute-1, fold-1-2 corresponds to Acute-2, and fold-1-3 corresponds to Acute-3. These folds have fold angles  $\theta_f$  between 0° and 90°. It remains for the ISG to complete the differentiation of situations in the top-level fold class that have fold angles in the interval (90°, 180°).

This is accomplished by randomly selecting a fold angle outside the fold-1 class and creating a new child of the fold class (*i.e.*, fold-2) with this value of  $\theta_f$ . Suppose this fold angle is 100°. An instance (*i.e.*, fold-2-1) of fold-2 is also created that has a specific orientation  $\theta_o$  between 0° and 90°. This branch of the fold tree is extended by the ISG into the branch of Obtuse folds in the same way as the original branch shown in Figure 6 was extended into the branch of Acute folds. It turns out that this new branch includes situations having fold angles between 90° and 180°.

A third child of the fold class (*i.e.*, fold-3) is created having the remaining unexplored fold angle  $\theta_f = 90^\circ$ . An instance (*i.e.*, fold-3-1) of fold-3 is also created with a specific orientation between 0° and 90°. The ISG extends this branch of fold situations into the branch of Right folds shown in Figure 3.

At this point the ISG has finished partitioning the initial set of fold situations. The output of the algorithm is the tree structure shown in Figure 3; except that the progeny of fold have labels of the form fold-k, and the progeny of each of these children have labels of the form fold-k-l.

## 5 Conclusion

#### 5.1 Summary

We have presented the ISG as a domain-independent weak method that exploits error propagation information to backpropagate a given partition of manifestations into a partition of interesting situations. Each interesting situation-manifestation pair can be used by the rule generator to synthesize an interpretation rule of the form manifestation  $\Rightarrow$  situation.

The ISG takes the most specific situation classes specified in a domain model and further differentiates them into equivalence classes that correspond to distinct manifestations. In this sense, it can help to 'complete' a domain model. If each of the given situation classes already maps into a single manifestation, the ISG makes no further divisions in the situation class hierarchy. Thus, the ISG picks up where the domain theory specifier leaves off and continues to differentiate the most specific situation classes until each of them corresponds to a single manifestation.

When a complete partitioning of the situations already exists, the ISG can be used to *verify* its correctness and completeness. In particular, the output of the ISG might be used to look for cases where two or more situation classes correspond to the same manifestation. In this way, the ISG could assist in focussing attention on those parts of the domain-model that require additional specification in order to be unambiguously detectable.

### 5.2 Current Status

The ISG can successfully differentiate folds with orientations between 0° and 90° and fold angles between 0° and 180° into 3 equivalence classes. It discovers that: (i) acute folds (those with fold angles less than 90°) produce red/blue, red/blue/red, and blue/red patterns; (ii) right angle folds (those with fold angles equal

to 90°) produce red/blue and blue/red patterns; and, *(iii) obtuse* folds (those with fold angles greater than 90°) produce red/blue, blue, and blue/red patterns. In another test of the algorithm, the ISG successfully differentiated folds with orientations between 90° and 180° and fold angles between 0° and 180°.

In its current form, the ISG program is implicitly tied to the fold domain. It cannot be applied to other problems, because its inputs (see Table 1) cannot be explicitly specified. For example, when applied to the fold problem, the program assumes the specific manifestation partition that distinguishes, for example, R/B from B/R patterns. The program cannot be run with a manifestation partition that only distinguishes, for example, composite patterns based on the number of their constituent primitive patterns so that R/B and B/R both belong to the same two-primitive-pattern equivalence class.

#### **5.3** Limitations

The ISG starts with error propagation information that describes how local changes in situation parameters affect manifestation parameters. During forward and backward error propagation, it uses this information to extrapolate the value of the manifestation parameters that result from larger changes in the value of situation parameters, and vice versa. This information is the basis for determining an equivalence class of situations that produce equivalent manifestations. The extrapolation is correct only in those cases where the situation and manifestation parameters are linearly related. Thus, the accuracy of the situation partition produced by the ISG depends upon the extent to which manifestation parameter values are linear functions of situation parameter values in the context of backpropagating each manifestation equivalence class.

The ISG is exponential in the number of undifferentiated situation parameters. That is, it is  $O(P^N)$ ; where P is the average number of intervals into which a situation parameter domain is partitioned, and N is the number of situation parameters. This result implies that the ISG can usefully be employed as a way of elaborating an almost-complete domain theory. It is not a practical way of partitioning situation classes that have more than a few undifferentiated parameters.

#### 5.4 Future Plans

We plan to isolate the ISG algorithm from the context of the folding problem. Once this is done, more of the ISG's inputs can be explicitly specified and independently varied. For example, we plan to experiment with the ISG by checking the results of varying (i) the order of situation parameters, (ii) the definitions of manifestation partitions, and (iii) the content of the domain theory that maps situations into manifestations.

At this time, the ISG program can only handle situations involving 2 parameters (e.g., folds with the parameters orientation and fold angle). After generalizing the program to handle any number of situation parameters, we plan to reapply it to the fold problem by adding another situation parameter—the orientation of the borehole. In this paper, we assumed the borehole had a vertical orientation.

As a weak-method, the ISG suffers from the limitations mentioned in Section 5.3. We need to better understand the extent to which these limitations constrain its utility. In particular, more experience and analysis is required in order to find out whether the ISG always terminates its search for interesting situations, and if it doesn't—under what conditions would one expect to observe divergent behavior. We need to have a better understanding of how the local nature of error propagation information compromises the accuracy with which the ISG partitions situation classes. If this will usually be the case, then we need to know whether it will be possible to and under what circumstances can we recover from such initial partitioning errors.

# References

- [1] Schlumberger Dipmeter Interpretation. Schlumberger Limited, New York, 1981.
- [2] B.G. Buchanan, T.M. Mitchell, R.G. Smith, and C.R. Johnson. Models of Learning Systems. Encyclopedia of Computer Science and Technology, 11:24-51, 1978. also Stanford Report STAN-CS-79-692.
- [3] B.G. Buchanan and E.H. Shortliffe. Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Addison-Wesley, Reading, MA, 1984.
- [4] R.M. Keller. Learning by Re-expressing Concepts for Efficient Recognition. In Proceedings of the National Conference on Artificial Intelligence, pages 182-186, Morgan Kaufmann, Washington, D.C., 1983.
- [5] Douglas B. Lenat. AM: Discovery in Mathematics as Heuristic Search. In Knowledge-Based Systems in Artificial Intelligence, chapter 1, pages 1-225, McGraw-Hill International Book Company, New York, 1982.
- [6] T.M. Mitchell. Learning and Problem Solving. In Proceedings of the Eighth International Joint Conference on Artificial Intelligence, pages 1139–1151, August 1983.
- T.M. Mitchell, S. Mahadevan, and L.I. Steinberg. LEAP: A Learning Apprentice System for VLSI Design. LCSR Technical Report 64, Rutgers University Department of Computer Science, New Brunswick, NJ, 1985.
- [8] T.M. Mitchell, P.E. Utgoff, and R.B. Banerjii. Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics. In Machine Learning, chapter 6, Tioga, Palo Alto, CA, 1983.
- [9] D.J. Mostow. Machine Transformation of Advice into a Heuristic Search Procedure. In Machine Learning, chapter 12, Tioga, Pala Alto, CA, 1983.
- [10] R. G. Smith and R. L. Young. The Design of the Dipmeter Advisor System. In Proceedings of the ACM Annual Conference, pages 15-23, ACM, New York, October 1984.
- [11] R.G. Smith, H.A. Winston, T.M. Mitchell, and B.G. Buchanan. Representation and Use of Explicit Justifications for Knowledge Base Refinement. In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, pages 673-680, August 1985.
- [12] W.R. Swartout. XPLAIN: A System for Creating and Explaining Expert Consulting Programs. Artificial Intelligence, 21:285-325, 1983.
- [13] H.A. Winston, R.G. Smith, M. Kleyn, T.M. Mitchell, and B.G. Buchanan. Learning Apprentice Systems Research at Schlumberger. In Machine Learning: A Guide to Current Research, Kluwer Academic Publishers, Norwell, MA, 1986.

Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov., 1986

# Mapping Cognitive Demands and Activities in Complex Problem Solving Worlds

David D. Woods

# Westinghouse Research & Development Center Pittsburgh, PA 15235 USA

Erik Hollnagel

Computer Resourses International Copenhagen, Denmark

### Knowledge Acquisition for Knowledge-Based Systems Workshop

Banff, Canada; November, 1986

## Building a Cognitive Description of a Complex World

Tool builders have focused, not improperly, on tool building — how to build better performing machine problem solvers, where the implicit model is a human expert solving a problem in isolation. A critical task then for the designer working in this paradigm is to collect human knowledge for computerization in the stand alone machine problem solver.

But tool use involves more. Building systems that are "good" problem solvers in isolation does not guarantee high performance in actual work contexts where the performance of the joint person-machine system is the relevant criterion. The key to the effective application of computational technology is to conceive, model, design, and evaluate the joint human-machine cognitive system (Hollnagel & Woods, 1983). Like Gestalt principles in perception, a decision system is not merely the sum of its parts, human and machine. The configuration or organization of the human and

machine components is a critical determinant of the performance of the system as a whole (e.g., Sorkin & Woods, 1985). The joint cognitive system paradigm (Woods, 1986) demands a problem-driven, rather than technology-driven, approach where the requirements and bottlenecks in cognitive task performance drive the development of tools to support the human problem solver.

In this paper we describe an approach to understand the cognitive activities performed by joint human-machine cognitive systems. The real impediment to effective knowledge acquisition is the lack of an adequate language to describe cognitive activities in particular domains — what are the cognitive implications of some application's task demands and of the aids and interfaces available to the people in the system; how do people behave/perform in the cognitive situations defined by these demands and tools. Because this independent cognitive description has been missing, an uneasy mixture of other types of description of a complex situation has been substituted — descriptions in terms of the application itself, of the implementation technology of the interfaces/aids, of the user's physical activities or user psychometrics.

We describe one approach to provide an independent cognitive description of complex situations that can be used to understand the sources of both good and poor performance, i.e., the cognitive problems to be solved or challenges to be met. The results from this analysis help to define the kind of solutions (tools) that are needed for successful performance, and the results can be deployed in many possible ways -- exploration training worlds, graphic interfaces, decision aids, machine problem solvers (c.f., e.g., Roth et al., 1986 and Bennett et al., 1986). The method can address existing joint cognitive systems in order to identify deficiencies that cognitive system redesign can correct and prospective joint cognitive systems as a design tool during the allocation of cognitive tasks and the development of an effective joint architecture (e.g., Pew et al., 1986).

To build a cognitive description of a complex world the first hurdle is to escape from the language of the application and to characterize the kinds of cognitive demands that arise in the course of performing domain tasks (Clancey, 1985). To carry out this translation we start from the constraints imposed by the design and proper function of the system in question (Lind, 1981) via a kind of goal directed analysis to map a functional representation of the domain (e.g., Davis, 1984; Rasmussen, 1983; 1986). The resulting functional representation serves as a framework and guide to identify — what kinds of problem solving situations can arise in this world? what must people know and how they must use that

42-1

2

knowledge to solve these problems (what are the difficult cognitive demands)? how do people actually respond in these types of problem solving situations (where do breakdowns such as fixation effects occur, Woods, 1984)?

Thus, the approach combines an analysis of the domain to determine psychological, particularly cognitive, demands and a psychological analysis of human, especially problem solving, performance given those demands. One can think of problem solving situations as composed of three basic elements: the world to be acted on, the agent who acts on the world, and the representation of the world utilized by the problem solving agent. To build a cognitive description of a world we need to capture the relationships between the cognitive demands of domain tasks, how the available representation of the world effects problem solver's information processing activities and problem solving performance, and the characteristics of the problem solver (cf., Woods, in preparation). There is an intimate relationship between these three elements so that cognitive description is at once a psychological description and a domain description. In our particular approach to describe cognitive activities we begin with and build on an analysis of the cognitive demands imposed by the requirements of the domain. This variant of the general approach, which we will focus on here, has been developed and refined over a number of years in the context of thermodynamic systems, e.g., problem solving during power plant emergencies (Lind, 1981; Rasmussen, 1983; Woods & Roth, 1986). It has also been applied on a smaller scale to electrical distribution systems, data communication network management, and logistics maintenance systems. A variant on this technique has been applied to emergency management, CAD/CAM, office systems and library systems (Rasmussen, 1986). A similar technique has been used to aid designers of aerospace cockpits (Pew et al., 1986). Embrey & Humphreys (1985) and Boel & Daniellou (1984) have begun to developed methods to develop goal directed structures starting from analyses and interviews of domain problem solvers rather than from knowledge about proper system function (in part to map errors in people's knowledge about system structure and function).

The approach helps the cognitive technologist to ask meaningful questions and provides a framework to integrate the answers acquired from diverse sources: some questions relate to domain technical knowledge so the knowledge acquisition problem is finding the right specialist to talk to or to point you to the right documents/analyses; sometimes the path is look empirically at how the problem is solved, e.g., critical incident analysis (Pew et al., 1981) or putting the problem solver in the situation of interest or simulations of it either formally (experiments) or informally; and sometimes the path is simply to ask the person who performs

3

- - 1

the task. All of these specific techniques are potentially useful; the choice depends on the resources and constraints of the knowledge acquisition enterprise and on where information about the domain in question resides.

The result of using this approach is a characterization of the kinds of problems to be solved in the domain and the relevant psychology of human performance in those situations. We will describe the approach primarily in the context the cognitive activities that arise in what we call disturbance management tasks (e.g., for problem solving in power plant emergencies). The approach was developed and has been used in for several efforts to provide support systems to improve joint cognitive system performance — in training systems that utilize the Steamer methodology (Bennett et al., 1986), in information management to aid problem formulation (Woods et al., 1986), in graphic representations to aid situation assessment (Woods, in preparation).

#### Mapping the Problem Space: Goal-Directed Representation

The paper is addressed to complex worlds, not because of limited applicability, but because complex worlds are the strongest tests of a methods effectiveness. Woods & Roth (1986) describes four dimensions that contribute to the complexity of problem solving: dynamism, the number and interactions between parts, uncertainty and risk, multiple agents. A domain's position (or the position of an application within a domain) along these four dimensions begins the description of its cognitive characteristics. The approach has been utilized most extensively in a world that is high on all four of these dimensions - power plant emergencies and is therefore well suited for dynamic worlds where large numbers of parts interact, where multiple failures can occur, and where the problem solver must cope with consequences of disturbances as well as repair causes of disturbances (disturbance management tasks). The technique for goal-directed representation is designed to map the relationships between parts, how the parts work, how evidence testifies to the state of these parts, and how each can change. This serves as a framework to describe the cognitive situations that arise in the course of carrying out domain tasks.

The mapping of cognitive demands begins with a goal-directed analysis that consists of defining the units of description (goals) and the relationships between those units that are appropriate for domain tasks (cf., Warfield, 1973; Lind, 1981; Pearl et al., 1982; Rasmussen, 1983 for descriptions of this type of analysis). A goal-directed

4

representation is constructed by structuring domain tasks in terms of the goals to be accomplished, the relationships between goals (e.g., causal relations, constraints) and the means to achieve goals (how a process functions, options, pre-conditions, post-conditions).

The basic unit of description is a goal-means inter-relationship. A function (the means) provides some commodity (the goal) that is needed or required in order for a higher level function to work properly. A function consists of the set of processes that affect this goal. Processes that make up a function, in turn, need to be supplied with commodities by lower level functions in order to work properly themselves (Figure 1a).<sup>1</sup>

- Goal: What the function should provide; some value of some commodity, e.g., the amount of material within control volume A (a reservoir) greater than Y.
- Function: The set of processes that provide the commodity and knowledge about how they work to achieve this goal, e.g., a balance of material inflow and outflow processes around the reservoir). A process consists of a set of alternative versions and their inter-relationships with respect to the goal (e.g., mutually exclusive, simultaneously active) and a model, in terms of functional elements, of how the process effects the goal; the functional elements of a model can themselves be processes (subprocesses) which possess models and alternatives.
- Requirements: What is needed for a process to be able to work properly if it is demanded, e.g., a source of material is needed for an inflow process to be capable of functioning (source reservoir level greater than Y). A requirement serves as a goal for a lower order function, e.g., the set of processes that can provide material to replenish the source reservoir. Requirements can occur at any level of decomposition of a process - the entire process, alternatives, a model functional element, or alternative versions of a functional element.



<sup>&</sup>lt;sup>1</sup>All of the specific examples used to illustrate the technique come from thermodynamic systems, in particular, emergency operations in nuclear power plants. The technique is general to any large complex system including logistics maintenance systems, power plants, chemical processes, air traffic control, flightdecks, medicine, data communications network management, and electrical distribution centers.

To see how this network is built consider a heat exchanger. Williams, Hollan & Stevens (1983) examined the mental models average people have of how a heat exchanger functions. Here our objective is to explore the representation an expert should have of a particular heat exchanger in a specific system context. Let us begin with the assumption that the designer of a complex thermodynamic system such as a power plant requires that the temperature within some control volume stay below some limit value. He or she then designs a process or processes, in this case a heat exchanger, to achieve this goal. In principle there are many aspects of the thermodynamics of a heat exchanger that the designer could use to control the temperature of interest -- the amount of mass, the rate of circulation, the kind of mass (different materials have different heat transfer properties), or the energy of the mass on either the primary or secondary side of the heat exchanger. In practice the designer chooses to use only some of these possibilities to control temperature in a particular case and the remainder are fixed at some value or range of values as a part of the environment of the heat exchange/temperature control process. For example, the primary side mass and circulation are fixed and temperature control is achieved through variations in the secondary side varibles in commercial electric power plants. These decisions define what is included in the temperature control process, how this particular process works, and what are the requirements needed for the process to be able to work properly (the primary side mass and circulation variables in this example). The variables that are fixed as features of the environment define other goals which must be achieved. For example, a function is needed to keep the amount of mass on the primary side of the heat exchanger above the minimum value necessary for proper heat transfer; namely a balance of material inflow and outflow procesess around the primary side of the heat exchanger. These judgements about what constitutes the "system" under consideration versus exogenous factors are often required when decomposing complex systems (e.g., Iwasaki & Simon, 1986).<sup>2</sup> Note that the term goal here is used to indicate domain goals, not the current objective of the problem solver.

This analysis can capture the interconnections between goals. Different functions can impose criteria to be achieved on a single commodity (Figure 1b). This means that breakdowns in the lower order function can result in violations of multiple criteria placed on a single commodity and therefore the disturbance can propagate

11

<sup>&</sup>lt;sup>2</sup>Note that in the case of an existing design this technique is a post hoc rationalization of the design and not a psychological description of the design process. The technique can also be used as a design evolves to aid design decision making.

in multiple directions in the functional topology of the domain. Second, a process may affect more than one goal, i.e., participate in several functions (Figure 1c). For example, a mass balance process will affect the goal of system material inventory, but because of thermodynamics, it also affects goals on system pressure and temperature. Each goal specifies a different reason to be interested in a process or a context in which to view that process. Goals can also be interconnected because of shared physical equipment, for example, a chemical carried by water when inventory of both materials is relevant. Thus, the goal-directed analysis can capture the kinds and extent of <u>inter-goal constraints</u> or side effects that exist in a domain: given a goal-of-interest, what constraints imposed by related goals govern the operation of a process to achieve the goal-of-interest.

A function is mapped by specifying the set of processes that can effect the goal in question. Each process is modeled qualitatively to represent how it works to provide a goal by decomposing the process into its functional elements and their interrelationships (see Lind, 1981 for one grammar of decomposition for thermodynamic For example, a material inflow process to a reservoir can be processes). decomposed into three functional elements: a source of material, a transport mechanism, and points of entry. Another part of decomposition is determining what if any are the alternatives for each process or functional element within a process (i.e., the redundancy and diversity in the system) and what is their interrelationship, such as mutually exclusive, simultaneously capable of effecting the goal, large versus small capability. For example, there may be multiple, mutually exclusive inflow processes to deliver the same material to the same destination that vary in how much or how quickly they can affect the goal commodity. Functional elements can continue to be decomposed into alternatives and models consisting of functional elements as is required to capture how the system being analyzed works. Thus, functions consist of processes, each of which has a model and alternatives, and the functional elements of a model can themselves be processes (subprocesses) Requirements can occur at any level of which possess models and alternatives. decomposition of a process - some value of a commodity may be needed in order for the entire process to function properly, for an alternative version of the process, for a functional element of the process's model, or for alternative versions of a functional element. Decomposing how processes effect goals is a kind of qualitative modeling (Bobrow, 1985) which in this case exists inside a knowledge representation network.

The goal-means network is a canonical description of the domain. It is a technically accurate description but not strictly normative because alternative, technically

7

42-6

accurate decompositions can exist. For example, a material process can be described as an inflow-outflow balance which can be decomposed into alternative inflow and outflow subprocesses which consist of source-transport and transport-sink functional elements, respectively. If some of the outflow sinks double as sources for inflow, then an alternative description would be to decompose the process into multiple recirculation versus injection processes.

The network that results from the goal-directed analysis has several interesting properties (Figure 2 is a portion of the network for the primary system thermodynamics in a nuclear power plant. First, moving up through the network defines reasons and the consequences that can result from disturbances if they are not checked. Moving down through the network defines causal chains and maps the routes for analysis of correctable causes for disturbances. Second, because the representation is not based on how the system fails (root cause faults) but on the system should function normally, it does not depend on complete enumeration of diagnostic (fault) categories or possible incidents. It represents a fault in terms of the disturbances (departures from normal functioning) produced by the fault and how those disturbances propagate over time (a disturbance chain). Multiple failures are represented by their respective disturbance chains and by how those disturbances interact. Thus, particular scenarios including  $\mathbf{their}$ temporal characteristics are represented in terms of the disturbances associated with the initiating fault and how those disturbances propagate over time and interact with the disturbances generated by other failures.

The network provides a framework to discover the kinds of problems that can arise, to define the kinds of information processing requirements, and to provide a conceptual foundation for investigations of actual problem solving performance.

### **Cognitive Demands**

Now we have a framework with which to capture the kinds of cognitive demands and situations that can arise in a particular domain which the problem solving agent must be able to cope with. At one level cognitive demands involve collecting and integrating available evidence to answer the questions about plant state (evidential uncertainty, data sampling, situation assessment). A second layer of cognitive demands occurs across units in the goal-means structure — given an assessment of system state (the mapping of available evidence into unit/attribute states at some point in a scenario), what cognitive situations arise in the choice of

8

a response including diagnosis, problem formulation (to choose the goal that should be focused on at that time in the problem solving process), plan generation or selection, choice under uncertainty and risk, plan monitoring and adaptation.

#### Cognitive Demands I: Evidence Utilization

••••

One kind of demand is to determine the state of the system or parts of the system. There are evidence processing requirements associated with each unit in the goal-means network that must be carried out to accurately specify the state of the system. These requirements can be determined by considering the monitoring and control activities that are needed to maintain goals. The particular questions that the person investigating the domain needs to answer are: what evidence can be used to determine (a) is a goal satisfied? (b) what constraints imposed by other goals are active? (c) is a process disturbed which depends on the questions should a process be active and is a process active? (d) can a process work if demanded? (e) how to initiate, tune, transfer or terminate processes. Figure 3 shows how these cognitive activities map into the structure of the goal-means network.

The goal directed framework specifies the <u>units</u> of description for situation assessment (independent of particular situations). The above categories specify the attributes of those units, e.g., a process may be inactive-available, inactiveunavailable, active-should be active, active performing correctly, etc. These unit: attribute categories define the situation assessment space; that is, they specify the set of questions that must be answered or what judgements must be made in order to characterize system state. The plant data that are or could be available form the raw material from which answers to these questions can be built. The problem solving agent must be able to collect and integrate data to answer these <u>questions about plant state</u>. For example, what parameters, targets, and limits indicate that a goal is satisfied or what data (e.g., tank levels, valve positions, pump status, flow rates) indicate that the function performed by a process is working correctly (e.g., a material flow process consisting of sources and transport mechanisms). In effect, a particular piece of plant data signals multiple messages or interpretations. At a high level, it signifies or activates a category in the situation assessment, e.g., x > y signifies a violation of goal A; at another level, the datum signifies that, of the set of things that represent violations of goal A, the particular signal is x > y.

Uncertainty is one of the dimensions of the complexity of problem worlds. Uncertainty in how evidence testifies to the state of units of the goal-means structure occurs because a given datum may or may not be an accurate

9

measurement/entry (e.g., sensor failure, misentry, overdue update on manual entry). Uncertainty also arises because a given datum may be evidence with respect to the state of several units in the goal-means network, in other words, there is a one to many data to unit mapping (Figure 6). In this approach, this form of uncertainty is represented explicitly by multiple evidential links from datum to unit. This is needed to be able to handle the flexibility with which experts can solve non-routine cases, to handle revisions of problem formulation especially in dynamic worlds, and to handle the possibility of multiple failures.

The problem solving agent's ability to meet these evidence processing demands is a joint function of several factors. First, the cognitive demands imposed by the domain can vary -- how much collection and integration of data is needed to answer the situation assessment questions, the kinds of integration required, how the world can change over time (which affects, for example, data sampling rate requirements). Performance also depends on the kind of representation of the available data provided to the problem solver (independent of different task demands, problem solving performance is a function of the problem representation). Each of these factors in turns depends on the processing characteristics of the problem solving agent. A complete cognitive analysis must address the interaction of each of these factors; failure to do so has been one source of failures in the history of decision support (see Woods et al., 1986 for one example -- the history of the alarm problem in process control and the various attempts to treat it). Here we are only focusing on how to map the first of these factors; see Woods (1984; in preparation) and Cole (1986) for examinations of how variations in problem representation effect human performance. Demands on the collection and integration of data are one potential source or contributor to performance difficulties on domain tasks, and it is one location in the cognitive landscape where support systems or decision automation may be deployed.

#### Cognitive Demands II: Pragmatic Reasoning Situations

How can we use the goal-means network plus answers to the information processing demands as a problem space to specify the kinds of cognitive situations that can arise? Traditionally, cognitive situations are described in global categories like diagnosis and planning or in terms of elementary information processing tasks (Card, Moran & Newell, 1983; Johnson & Payne, 1985). But the goal-means problem space offers a mechanism to describe at an intermediate level the kinds of situations the problem solver may confront. The result can be thought of as pragmatic reasoning situations, after Cheng et al. (1986) and Holland et al. (1986),

~~ E

in that the situations are abstracted from the specifics of the domain language and the reasoning involves knowledge of the things being reasoned about. The goalmeans problem space provides the knowledge about the relationships between units (e.g., goal, inter-goal constraints, process decompositions, requirements) and about the state of units (especially about desired/actual state mismatches) that can be used to describe the reasoning situations that arise for particular incidents in particular worlds.

For example, planning a response to a goal-of-interest should take into account the constraints imposed by related goals in order to avoid unintended effects (missing side effects is a typical error in complex systems; Dorner, 1983). For example, does the problem solver know that a requirement relation exists and is now relevant between units A and B (or a post-condition, goal, process alternative, inter-goal constraint, or other relationship)? If, for other reasons, A needs to be done, then B must first be satisfied; therefore, check if B is satisfied. If B is not satisfied, then A cannot or must not be done; therefore, do not even consider A with respect to the original reason you were interested in it or act to satisfy B so as to able to do A. If B is satisfied, then A can be done (e.g., Cheng & Holyoak, 1985). Similar reasoning processes can easily be seen for other relationships such as post-conditions where the problem solver is biased towards one alternative over another because of the negative post-conditions associated with one of the paths.

To illustrate a few of the kinds of pragmatic reasoning situations that can arise and be described in terms of the goal-means space, consider one of the four dimensions that contribute to the complexity of a problem domain: the level of interconnections between parts. When interconnection are extensive, actions involving one part can often have side effects in addition to the intended effect; a single fault can have consequences for multiple parts; independent faults can have consequences for the same part. Extensive inter-relationships between parts also introduce complexity because they can produce "effects at a distance" (Boel & Daniellou, 1984 contains one example); that is, a fault in one place can produce disturbances at other distant locations (given either a physical or functional distance metric). Automation (multiple agents) is a potent source of effects at a distance. Highly coupled systems produce the opportunity for situations to arise where there are competing goals. Competing goals can take two forms. One is a choice involving goal sacrifices where the tradeoff is between repairing or replacing one means to achieve the primary goal (can I repair it? how much time is available to attempt repair?), and an alternative means that entails as postconditions the sacrifice of other goals in order to meet the primary goal. A second

case is multiple mutually constrained goals where two or more goals must be simultaneously satisfied in circumstances where the means to achieve each constrains how the others can be achieved.

When a large number of interconnected parts is combined with another dimension of complexity, dynamism, the disturbance management cognitive situation arises. In this situation the problem solver must cope with the consequences of disturbances in the short run as well as diagnosis and treat the underlying fault (get the plane into stable flight, then figure out how to restart the failed engine). The problem solver in this kind of problem situation first becomes aware of a disturbance (actual/desired state mismatch). The problem solver has several possible strategic responses: attempt to adjust the disturbed process, find and correct, if possible, what produced the disturbance (diagnosis), respond to cope with the effects of the disturbance (if there is insufficient time to repair or because of an inability to repair or replace the affected process).

In disturbance management, various pragmatic reasoning patterns are relevant, e.g., if there are no pressing consequences of a disturbance and no more urgent disturbances on the agenda, then try to find a correctable cause. Note that knowledge about the time response of various processes is often an important element in disturbance management, e.g., the time available until a disturbance propagates from one part to another, responses to buy more time before consequences propagate, the time it will take to adjust or repair a process. Also, judgments about <u>prospective</u> solutions can play a significant role -- will I be able to adjust or repair the process (implying knowledge of why it failed) before undesirable consequences occur.

Figure 4 illustrates a case of competing goals in terms of the goal-means structure that is abstracted from a real incident (NUREG-1154, 1985 and see Woods & Roth, 1986 for the cognitive analysis) and from multiple simulated incidents. One of two alternatives within a function was disturbed. The alternative process (B) needed to be activated to prevent consequence propagation (which would occur in about 30 minutes), but activation of the alternative would have a significant negative impact with respect to a second relatively less important goal. Another path open to the problem solvers was to try to diagnosis and correct the disturbance in alternative A. Actual problem solvers chose to try to repair A because they judged that they would be able to do this within the time available (in part because they saw the disturbance as a misstart rather than as a failure). Although they needed to be prepared to adapt their response if conditions changed

which increased the urgency of preventing consequence propagation or if repair efforts met difficulties.

The cognitive situation of diagnosis of correctable causes is represented in the goalmeans space in terms of working down through a function's decomposition and requirement links. Figure 5 illustrates the causal chain for a case where the problem solver begins with a goal violation -- reservoir level is low. The first step illustrates one difficult kind of diagnostic situation where multiple processes are simultaneously capable of effecting the goal. Thus, any one is capable of being the source of the disturbance. This situation is particularly difficult when multiple processes are active because the effect of one contributor partially occludes the effect of the other. For example, when one is identified as a contributor to the disturbance, the diagnostic search at that level can be terminated prematurely. Figure 5 continues a trace of the causal chain; the problem solver terminates the search when a correctable item is located. This depends on the purpose of the problem solver and on the action set available to him or her.

#### Summary

We have described briefly an approach to map the cognitive demands imposed by the characteristics of a problem solving world and illustrated some examples of the descriptive power of this approach. The approach is part of an effort to develop a cognitive language of description that is independent of the language of the domain itself and the language of interface/decision support technology so that we can understand not only the cognitive demands imposed on a problem solver, but also the various shapes and guises of decision support systems and decision automation.

#### References

Alterman, R. An adaptive planner. In Proceedings of the AAAI. AAAI, 1986.

- Bennett, K., Woods, D., Roth, E. & Haley, P. Predictor displays for complex, dynamic tasks: A preliminary investigation. In Proceedings of the Human Factors Society. 30th Annual Meeting, 1986.
- Bobrow, D. G. (ed.). Qualitative Reasoning About Physical Systems. Cambridge, Mass.: The MIT Press, 1985.
- Boel, M. & Daniellou, F. Elements of Process Control Operator's Reasoning: Activity Planning and System and Process Response Times. In D. Whitfield (Ed.), Ergonomic Problems in Process Operations. Inst. Chem. Eng. Symp.

Ser. 90, 1984.

- Card, S. K., Moran, T. P. & Newell, A. The Psychology of Human-Computer Interaction. Hillsdale, N.J.: Lawrence Erlbaum Associates, 1983.
- Cheng, P. W. & Holyoak, K. J. Pragmatic reasoning schemas. Cognitive Psychology, 1985, 17, 391-416.
- Cheng, P. W., Holyoak, K., Nisbett, R. & Oliver, L. Pragmatic versus syntactic approaches to training deductive reasoning. *Cognitive Psychology*, 1986, 18, 293-328.
- Clancey, W. J. Heuristic Classification. Artificial Intelligence, December 1985, 27(3), 289-350.
- Cole, W. G. Medical Cognitive Graphics. In Chi'86 Conference Proceedings. ACM, 1986.
- Davis, R. Reasoning from first principles in electronic troubleshooting. International Journal of Man-Machine Studies, 1983, 19, 403-423.
- Dorner, D. Heuristics and cognition in complex systems. In R. Groner, M. Groner & W. F. Bischof (Eds.), *Methods of Heuristics*. Erlbaum, 1983.
- Embrey, D. & Humphreys, P. Support for decision making and problem solving in abnormal conditions in nuclear power plants. In *Knowledge Representation for Decision Support Systems*. Durham, England: International Federation for Information Processing, 1985.
- Holland, J. H. Escaping brittleness: The possibilities of general purpose machine learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.), Machine Learning: An Artificial Intelligence Approach. Los Altos, CA: Kaufmann, 1986.
- Holland, J. H., Holyoak, K. J., Nisbett, R. E. & Thagard, P. R. Induction: Processes of Inference, Learning, and Discovery. Cambridge, MA: MIT Press, 1986.
- Hollnagel, E. & Woods, D. D. Cognitive Systems Engineering: New wine in new bottles. International Journal of Man-Machine Studies, 1983, 18, 583-600.
- Iwasaki, Y. & Simon, H. A. Causality in device behavior. Artificial Intelligence, 1986, 29, 3-32.
- Johnson, E. & Payne, J. W. Effort and accuracy in choice. Management Sciences, 1985, x, x-x.
- Lind, M. The use of flow models for automated plant diagnosis. In J. Rasmussen & W. B. Rouse (Eds.), Human Detection and Diagnosis of System Failures. New York: Plenum Press, 1981.
- Pearl, J., Leal, A. & Saleh, J. GODDESS: A goal-directed decision structuring

14

system. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1982, PAMI-4, 250-262.

- Pew, R. W. et al. Cockpit Automation Technology (Tech. Rep. 6133). BBN Laboratories Incorporated, 1986.
- Pew, R. W., Miller, D. C. & Feehrer, C. E. Evaluation of Proposed Control Room Improvements Through Analysis of Critical Operator Decisions. Palo Alto, CA: Electric Power Research Institute, 1981. (NP-1982).
- Rasmussen, J. Skills, rules, and knowledge; signals, signs, and symbols; and other distinctions in human performance models. *IEEE Transactions on Systems*, Man, and Cybernetics, 1983, SMC-12, 257-266.
- Rasmussen, J. A Framework for Cognitive Task Analysis. In Hollnagel, E., Mancini, G. & Woods, D. D. (Eds.), Intelligent Decision Support. New York: Springer-Verlag, 1986.
- Rasmussen, J. A cognitive engineering approach to the modelling of decision making (Tech. Rep. Riso-M-2589). Riso National Laboratory, 1986.
- Roth, E., Woods, D. & Gallagher, J. Analysis of expertise in a dynamic control task. In Proceedings of the Human Factors Society. 30th Annual Meeting, 1986.
- Sorkin, R. D. & Woods, D. D. Systems with human monitors: A signal detection analysis. Human-Computer Interaction, 1985, 1, 49-75.
- Warfield, J. N. Intent structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 1973, SMC-3, 133-140.
- Williams, M. D., Hollan, J. D., & Stevens, A. L. Human reasoning about a simple physical system: a first pass. In D. Gentner & A. S. Stevens (Ed.), Mental models. Hillsdale, N.J.: Lawrence Erlbaum Associates, 1983.
- Woods, D. D. Visual Momentum: A concept to improve the cognitive coupling of person and computer. International Journal of Man-Machine Studies, 1984, 21, 229-244.
- Woods, D. D. Some results on operator performance in emergency events. In D. Whitfield (Ed.), Ergonomic Problems in Process Operations. Inst. Chem. Eng. Symp. Ser. 90, 1984.
- Woods, D. D. Paradigms for Intelligent Decision Support. In Hollnagel, E., Mancini, G. & Woods, D. D. (Eds.), Intelligent Decision Support in Process Environments. New York: Springer-Verlag, 1986.
- Woods, D. D., Elm, W. C. & Easter, J. R. The Disturbance Board Concept for Intelligent Support of Fault Management Tasks. In Proceedings of the International Topical Meeting on Advances in Human Factors in Nuclear Power., 1986.

16

Woods, D. D. The Design of Problem Representations. Manuscript in preparation.

Woods, D. D. & Roth, E. The Role of Cognitive Modeling in Nuclear Power Plant Personnel Activities: A Feasibility Study. Washington D. C.: U. S. Nuclear Regulatory Commission, 1986. (NUREG-CR-4532).



\*\*\*\*

Figure 1 (a): Structure of the goal-means network: the goals to be accomplished, the relationships between goals and the means to achieve goals.



Figure 1 (b): Different functions may place different requirements on a single goal (G1); or from a different point of view function C must satisfy multiple criteria on G1.



Figure 1 (c): Given G1 as goal-of-interest, Process 2 provides G1 and affects G2 (G2 is a side effect of, or a constraint on Process 2 if G1 is the object of interest and visa versa if G2 is the object of interest.

© 1985 Westinghouse Electric Corp.


- Figure 2. Simplified example of goal-means network for one part (primary system thermodynamics) of a nuclear power plant.
  - © 1985 Westinghouse Electric Corp.

1

42-17

\$





© 1985 Westinghouse Electric Corp.

42-18

\$

RELATIONSHIPS:

1. ALT A DISTURBED (should be active with respect to Goal 1, but is not)

2. ALT B ACTIVE RESULTS IN DISTURBANCE IN FUNCTION B

(should not be active with respect to Goal 2, but would be)

3. GOAL 1 > GOAL 2

## DECISION PROBLEM: ATTEMPT TO REPAIR ALT A BEFORE DISTURBANCE PROPAGATES TO GOAL 1 OR SWITCH TO ALT B AND ACCEPT TRADEOFF



## Figure 4. Illustration of disturbance management cognitive situation.

© 1985 Westinghouse Electric Corp.

42-19

\$



Figure 5. Illustration of diagnosis in terms of goal-means structure.

.

© 1985 Westinghouse Electric Corp.





© 1985 Westinghouse Electric Corp.

1

42-20

5