Reprinted from *Int. J. Man-Machine Studies* (1978) **10**, 285-294

A program structure for event-based speech synthesis by rules within a flexible segmental framework†

DAVID R. HILL
Man-Machine Systems Laboratory, Department of Computer Science, The University Calgary, Alberta, Canada T2N 1N4

(Received 27 May 1977)

A program structure based on recently developed techniques for operating system simulation has the required flexibility for use as a speech synthesis algorithm research framework. Synthesis is possible with less rigid time and frequency-component structure than with simpler schemes, and it allows much of the speech knowledge required for synthesis to be removed from the main driving structure and embodied as tables and procedures that may easily be modified or replaced. The program also meets real-time operation and memory-size constraints. The resulting view of speech structure, at the acoustic-segmental level, is that of time-ordered, perceptually relevant events, and is related to that used in the author's work on automatic speech pattern discrimination. The flexibility of the scheme for synthesis, and the excellent mutual independence of the many processes, with differing objectives, that must be run for realistic approximations to real speech variation, have proved a welcome release from earlier problems.

Introduction

The production of artificial speech, independently of any particular real utterance, has an ancient and respectable history. But only with the invention of the channel vocoder, and the adaption of its synthesizing terminal to keyboard operation, did the synthesis of connected speech on demand become practical. Even then it relied upon rules that were unformalized and relatively inaccessible within the heads of operators whose training took a year or more.

The invention of more easily controlled synthesizers that also modelled relevant aspects of human speech production, coupled with the development of improved control means for the vocoder-related synthesizers of the 1950's (such as the Haskins Laboratories Pattern Playback), led to very important advances in our knowledge of perceptually relevant speech structures and to the beginnings of speech synthesis by machine based upon a realistic set of formalized rules. At the same time computers were coming into widespread use and, during the 1960's, programs were written at a number of laboratories that embodied, in somewhat simple form, the knowledge required for speech synthesis by rules, as then known. The programs usually provided facilities for the manual entry of control parameter data for those aspects of speech that had not been formalized into sets of required rules, notably for rhythm and intonation (pitch variation) which comprise the chief suprasegmental features of speech. Since then, experiments have continued, using parameter data copied from real utterances together with improved and extended sets of rules for speech synthesis, in an attempt to improve the naturalness and intelligibility of machine synthesized speech. Those experiments with real speech data have shown decisively that the main limitations do not lie in the hardware synthesizers that are

†Original presented at 5th Man-Computer Communications Conference, Calgary, May 1977.

0020-7373/78/030285 + 10 $02.00/0

used. Speech that is indistinguishable from the original (except perhaps by trained experts) has been produced using parameter data copied from real speech (Holmes, 1973). It seems instead that the problems lie in the crudity and lack of sophistication of the rules and data used for speech synthesis. For a more detailed account of the context to the present work, the reader is referred to two review papers by the author (Hill, 1971, 1972).

Justification for, and orientation of the research

The history of computers over the last 30 years has shown a growing concern for easy, natural and effective ways of communicating with these increasingly powerful information processing machines. The combination of mini- and microcomputers plus easily controlled synthesizers with this growing demand for better means of communications between people and machines (especially the possibilities offered by telephone access) has led to an upsurge of interest in the development of speech synthesizing peripherals capable of translating text, delivered at normal character transmission rates, into spoken output. The alternative, using part of the resources of a larger system to drive speech synthesis hardware, is equally of interest.

As part of a continuing joint project in man-computer communication using speech, shared by the Department of Computer Science at the University of Calgary, and the Department of Electrical Engineering Science at the University of Essex (U.K.), experiments in speech synthesis have been carried out involving both engineering development on the basis of present knowledge, and basic research designed to shed light on some of the unsolved problems in the area. In particular, we have sought improvements in segmental synthesis; improvements in the specification of speech rhythm and pitch variation in terms of the assumed structure for these phenomena, as well as the assignment of specific patterns; and, as a result of early experiences in this work, improvements in the synthesis program structures used. It is with this latter class of improvements that the present paper is concerned.

The problems

One problem in speech synthesis by machine based on rules, assuming a reasonably sophisticated hardware synthesizer, is that the procedures that must be executed, and the parameters that must be computed, vary for different segments, whilst interactions frequently need to be allowed for across several segments. Also, our knowledge is still incomplete and, for research on the problems, considerable flexibility must be maintained whilst still requiring both an efficient enough program to generate speech in real-time for purposes of evaluation and experiment, and an economical enough program to fit within an economically viable hardware environment for marketplace development trials. There are other problems. Although speech is frequently treated as having independent segmental and suprasegmental aspects, the parameter variations arising from these two sources are likely to interact and/or need to be superimposed. Certainly cues to both segmental and suprasegmental aspects of speech are likely to be carried by the same parameter(s)—for example, pitch variation. Some effects may apply globally, but not equally throughout an utterance (for example, the rate of speech is linked with vowel reduction, but has a different effect on consonants). In all this, the possibly conflicting effects of a variety of different factors, represented by portions of the program, need to be

reconciled. Another type of problem is that involved in computing continuous changes for the synthesizer control parameters on the basis of a segmental input that is highly discrete. What is actually done is based upon what may be termed the "segmental assumption" that lurks in many approaches to speech synthesis by computer, and makes for changes in control parameters that begin and end both abruptly, and in synchronism, despite the obvious requirement not to do this. The segmental assumption considerably simplifies the computing problems, but is very likely an important cause of unnaturalness in machine generated speech, not necessarily just because of the effect at the segmental level, but also because of undesirable interactions with suprasegmental cues (for example, Hill & Reid, 1977). A third kind of problem is that involved in organizing the running of a complex heterarchical system and arranging the transfer of information between processes in the system. Finally, a fourth kind of problem arises from the need to achieve economy of processing time (to allow good speech in real-time) and economy of memory space (since economically viable applications imply that a single computer is not devoted to a single speech channel, unless it happens to be a microprocessor with limited memory).

Solution of synthesis problems by simplifying synthesizer controls

Some commercial ventures have been based on accepting the limitations of simple control apart from those devices that synthesize utterances from assemblies of pre-recorded speech or parameter tracks. The approach is exemplified by applications of the VOTRAX® speech synthesizer. This device, with the associated control programs, takes an uncompromisingly segmental view of speech structure; produces, with care, surprisingly good speech output directly from low-data-rate symbol-string input; and makes a virtue of necessity by matching the sophistication of the synthesizer to the simple control strategies assumed, which keeps its cost low. However, considerable phonetic knowledge is needed to compose the symbol strings required for arbitrary utterances, transitional cues in speech are approximated by arbitrary sequences of steady-state spectral chunks, and it is still necessary to solve the problem of pitch and duration assignment for the intonation and rhythm of connected utterances, with only rather crude facilities for control of these factors. Only one accent and speaker are possible, due to the fixed nature of the symbols in relation to their acoustic specification, and the device is incapable of implementing many segmental features found in natural speech. In the absence of more complete rules for the whole speech synthesis process, however, it is an excellent practical compromise, though it is unsuited to fundamental research.

The segmental assumption of speech synthesis is highly explicit in such a device. Acoustic changes occur all at once, at segment boundaries. This unreal assumption, and the associated assumption that all the cues relevant to the *identification* of a real speech segment lie between such arbitrary boundaries, is one that has led to many failures in speech recognition research. It is not unreasonable to suppose that it is one cause of poor intelligibility and unnaturalness in speech synthesized by rule. As previously noted, given the ability to vary synthesizer control parameters continuously and asynchronously, exceedingly good synthetic speech may be produced. The ultimate goal of our synthesis research is to reduce the specification of such natural variations to a reasonable set of rules and procedures, which requires a departure, at least for trials, from the segmental assumption.

Steps to solution based on a program structure allowing arbitrarily complex control strategies

The new program structure developed by the author for speech synthesis has three main divisions: (a) an interrupt handler (RTSO) to drive the synthesizer hardware; (b) a segmental synthesis part (SEGSYN) that takes data specifying the identity and duration of speech sounds, together with data specifying the variation in voicing frequency, and computes the data for RTSO; and (c) a suprasegmental part that translates the input text-strings, with special annotations and punctuation, into the data required by SEGSYN. Another paper being presented at this conference (Witten & Smith, 1977) considers some problems and solutions in the suprasegmental part so that only RTSO and SEGSYN are considered in this paper.

The present scheme was designed:

(a) to operate with less rigid attachment to the segmental time framework than simple schemes;
(b) to remove as much as possible of the speech knowledge content of the program from the program structure;
(c) to give the greatest degree of flexibility in choice of synthesis methods—especially allowing for the many varieties and spans of interaction effects, and providing for mutual independence of any procedures used, whilst still allowing total freedom for procedures operating at any level to influence or modify the results produced by other procedures at the same or different levels, or even to initiate arbitrary procedures;
(d) to provide for easy and natural control of the various processes; and
(e) to meet real-time operation and memory size constraints.

The current package represents a partial solution to these and other practical problems. It is, however, of considerable interest that the resulting view of speech, at the acoustic-segmental level, is that of time-ordered perceptually-relevant *events*—precisely the view of speech structure that has had to be adopted in the author's related work on automatic speech recognition for equally practical but different reasons. It is therefore desirable, before continuing with details of the synthesis program, to digress and briefly to examine the nature of the event-based view of speech as it arises in automatic speech recognition, as an aid to understanding the character and implications of an event-based view of speech structure.

In order to discriminate speech utterances into different classes in a manner that parallels some aspect of human speech recognition, an adequately general means of describing speech utterances in terms of perceptually relevant cues must be developed as a necessary part of the process. Certainly up until the late 1960's, and even continuing to some degree today, two major problems of generating such descriptions have been called "the segmentation problem" and "the time normalisation problem". The present author has suggested (Hill, 1971) that they are really different views of the same problem, arising from failure to handle the time aspect of speech structure adequately, and has developed an approach to automatic speech recognition that avoids these two problems in their classical form by using a description of speech based upon perceptually relevant *events*—time epochs in a variety of measurements of perceptually relevant speech cues. Such events, having no time extent, can meaningfully be ordered with respect to each other in time. An event might, for example, mark the beginning of a transition-from one frequency region to another—of a peak in the frequency spectrum, or perhaps the end of

some characteristic combination of spectral components after a comparatively long time. In recognizing an utterance, the most likely utterance is hypothesized on the basis of fragments of evidence, each of which comprises some time-ordered subset of all the events detected. This is analogous to the way the most likely overall molecular structure of an organic chemical compound may be hypothesized on the basis of fragments of evidence, each of which comprises the detection of some spatially ordered fragment of the original structure using standard mass-spectrometer analysis (Buchanan *et al*., 1969). The author's recognition system, currently under development at the University of Calgary, is based upon the event-based view of speech structure. It is a long-term project involving both hardware and software development, as well as fundamental speech research. However, the basic system hardware is almost complete, and an initial system is expected to be operational within a year, many of the ideas and subsystems having already been tested.

It would seem self-evident that, if a description of speech structure is adequate or, better still, superior for speech recognition, it should be adequate or possibly superior as a basis for speech synthesis, and that a unified approach to both problems should represent an encouraging step forward, allowing progress on one to give insight into solutions to problems for the other. For some time after the approach to describing structure for recognition purposes was developed, the equivalent approach to describing speech structure for synthesis remained tantalizingly obscure. It is now felt that there were two reasons for this. One reason lies in the previously mentioned segmental assumption of speech synthesis and the other in the fact that, until recently, the necessary computer programming techniques simply had not been developed or, at least, had not been seen to be relevant. There are certainly many other problems to solve before the goal of a completely unified and adequate description of speech for both recognition and synthesis is achieved, but the present coincidence of requirements for synthesis and recognition may represent an important step towards that goal, although the requirements arose for different practical reasons in each case, and were unrelated to that goal.

In order to provide the required heterarchical basis for organizing the calling of procedures and transfer of data between procedures, the two components RTSO and SEGSYN are each organized along the lines of an operating system simulation (MacDougall, 1970). The tasks that must run under SEGSYN, the segmental synthesis program, are those that interact with the suprasegmental part, compute the data required by the interrupt handler RTSO, organize the process of segmental synthesis, take account of special directives originating from the operator but passed along with the text string (for example, a directive to suspend some particular facility), and take care of errors and exceptions. The last mentioned facility includes providing "firewalls" to error propagation—an essential precaution in building a complex system with the express intention of updating and modifying it. Considerable programming effort was taken up foreseeing and providing for conditions that were "impossible" during correct operation, being independent of data input, and the effort turns up as protocols for communication, for example, rather than large amounts of code. The output data from SEGSYN comprises the real-time event list which drives the interrupt handler, RTSO, together with parameter rate-of-change data, stored in circular queues. The latter data are associated with the basic acoustic-parameter events that form the backbone of the synthesis strategy. This data structure not only implements an economical form of run-length coding for synthesizer control-parameter variation and makes for very simple organization of the driving components, but also allows for easy superposition of effects from different

sources and allows freedom to depart from any rigid time-structure across the collection of parameter changes. The interrupt handler naturally runs in real-time. Each tick of the realtime clock corresponds to an interrupt request for new parameter data from the synthesizer. Interrupts occur every ten milliseconds. As a result of a given interrupt, the real-time clock value is incremented; a new set of parameter values is calculated, based on the current rate-ofchange and the last values, and sent to the hardware; and then the real-time event-list is inspected to see if an event is due. If no event is due, the interrupt is dismissed. If one or more events are due, the required processes are run. Provision is made for the detection of overdue events which, in the present system, cause an error process to be run and the synthesis to be terminated at that point. For a basic acoustic parameter event and for each parameter specified by the event, change-of-rate-of-change information is picked up from the appropriate data queue and used to update the table used by the synthesizer control-parameter driver. The queue handling routines are set up to detect and fail-soft on a number of error conditions, and also operate with a refill threshold. This latter feature allows the processes that supply the realtime data to be restarted whilst there is still time to produce new data, without interrupting the synthesis, after having previously been put to sleep due to tack of room for the data they produce in the receiving queues. The principal benefit from this is that the queue sizes may be kept very small regardless of the utterance(s) to be synthesized. The explicit error handling means that bugs in the original program, as well as those arising from changes and additions later, are caught explicitly and contained.

The segmental synthesis program, SEGSYN, runs in pseudo-time—currently ten times faster than real-time, being limited only by the time taken to run the required processes. Events are taken from the pseudo-time event-list, in time order, and the pseudo-clock is advanced to each event time in turn. This clock, together with computed time delays, determines the times at which future events will be scheduled, both for the pseudo-time event list and for the real-time event list. Events in the pseudo-list have to do with the synthesis process and are quite distinct from the mainly acoustic-parameter events in the real-time list. Any pseudo-event may arbitrarily schedule events for both pseudo-time and real-time lists, and may also run other arbitrary processes. Thus each pseudo-event, in turn, becomes the controlling factor for synthesis, potentially determining the course of future events according to its own recipe. In this way the synthesis program comes close to writing itself, according to the changing requirements of the ongoing synthesis.

The processes that are run by each of the pseudo-events may be associated with individual acoustic segments, or may be chosen on the basis of dynamically varying control data, or may represent global conditions and effects, or maybe chosen according to basic synthesis strategies. Adding new events and processes concerned either with synthesis or with real-time operations merely requires the writing of an appropriate new procedure and the patching of a jump table, but does not increase the organizational complexity because control always resides with the current event, interactions between processes are achieved on the basis of superposition of effects, and all processes have read and write capabilities for both control data and output data. Similar changes to the old program were not only much less straightforward and much more complexity-creating, but also involved interaction with the large data buffers that were needed in that program—a fruitful source of bugs, involving pointers and *ad hoc* dynamic storage manipulations.

Events may be used for a variety of purposes. These include the generation of parameter control data, direct parameter control, communication, error handling, compu-

tation, and data retrieval or transmission. External control data may be passed along with the input text string on which the synthesis is based. Also, certain synthesis options may alternatively be selected using the computer console handswitches to facilitate interaction during development and experimentation. Provision is made for the pseudo-time process to "go to sleep", either when it runs out of input data, or when it fills any of its output buffers (the real-time list and queues used by RTSO). It is woken again when new input is available or when the real-time queue processor detects a refill level, as noted above. As a result of this strategy, other processes (including, for example, programs to handle the suprasegmental aspects of the synthesis, or programs to carry out text-to-phoneme conversion) may run concurrently, leading to a further level of operation for the heterarchy of processes concerned with synthesis as well as permitting the entire synthesis package to act peripherally to some main process that requires speech output but which need know nothing of the means used to achieve it.

Discussion, including some details of implementation and use

The current version of the new synthesis program has few of the possible extensions so that the output produced apparently differs little from that produced by the earlier program. However, despite its facility for more complex control strategies, it occupies only one half the amount of core (approximately 1500 words) occupied by that program, both being coded for the same 12-bit word-length minicomputer, and the data space for output is negligible by comparison. One important departure concerns the separation of pitch specification from segment specification, which has considerably eased the problem of specifying the pitch pattern for an utterance as well as the problem of computing the parameter data for the pitch channel. At the same time, the specification of segment duration has been rationalized on linguistic criteria that suppose a given segment of speech relates to a posture of the articulators, and begins with the release from the previous posture, and ends with the release to the next posture. Thus each segment comprises the transition to, and holding of, some notional target posture defined, at the lowest level, in terms of table values. As a result, the input to SEGSYN comprises two queues: one queue (INT) holds posture codes and durations, the other (PIT) holds pitch targets and target times. The data for these queues are supplied by the suprasegmental part of the program. When the data are available. SEGSYN is woken and, as part of the initialization, an NPOS event is scheduled which obtains the first posture code with its associated duration, schedules a further NPOS event for an appropriate time, sets up the table data for the posture just obtained, and schedules an RTIC event which "articulates" to produce real-time parameter data. Other events that NPOS may schedule currently include the EOU (end of utterance) event, or the special pitch event (SPIT) which is required if, for some reason, the duration of the utterance has been changed from that planned by the suprasegmental part of the package. NPOS also runs processes associated with the control codes that may be passed along with the posture codes in the INT queue, and these processes may be for all sorts of purposes, being called via a jump table.

As a further part of the initialization, a PICH event is scheduled. This event is concerned with the generation of the real-time data for basic pitch parameter events, as required to generate the basic intonation of utterances. It retrieves target value and target time data from the input PIT array and schedules the next PICH event as well as setting up the data for RTSO. The earlier synthesis program associated pitch specification with

the segmental data, which led to several problems. First, there necessarily had to be a time reference point within each segment to which pitch targets could be related. Deviation from this reference point had to be specified, along with the pitch target, and attached to particular segment specifiers. A second problem was that the density of pitch targets varied, depending on the intonation contour being implemented and the portion of the utterance being computed, so that it was not uncommon to require more pitch targets in a region of speech than there were segment specifications to carry them. Two solutions were tried. One solution involved the use of large displacement factors, so that pitch target values from other segments could be displaced into the region of interest while the other involved allowing several pitch specifications per segment. Both were unsatisfactory being (a) awkward to compute as they required variable amounts of back-tracking in the pitch parameter computations and/or extra program code to handle three varieties of pitch target per segment; and (b) unsuited to good interaction because the data for rhythm, intonation and segment specification were all confounded in one complex coding scheme that was not easy to interpret. The current scheme simplifies the specification, computation and interpretation. In particular, as many pitch specifications as are required may be placed anywhere (or as few) without any effect on the complexity of the computation, without any back-tracking, and without any loss of clarity (for the user) as to what is going on. Furthermore, a given utterance may have the intonation contour changed completely, without any editing of the utterance file, simply by calling a different contour from a table, or computing one by rule, and scaling it to the key time points of an utterance, prior to inserting it into the pitch data array.

The articulation event, RTIC, is the source of all basic acoustic-parameter events in the realtime list, and of all the basic change-of-rate-of-change data in the real-time queues, except for pitch data and events. Since the input data are discrete (the target values and posture durations passed by NPOS) it would appear that the segmental assumption is basic to its operation. It is true that an initial time framework, not unlike that of the earlier program, is set up on the basis of the data passed. Thus there are four major event times (or MET's): (a) the time at which the last steady state ends or the release to the current posture begins (ESS/BT); (b) the time at which the current steady state begins (BSS); (c) the time at which the current target-values are most nearly achieved (SST); and (d) the time at which the release to the next posture will begin (ESS/BT). Each call to RTIC computes a new set of three of these (the last three), according to the types and combinations of postures present, and then uses the first three. The parameter values to be achieved at each MET are computed on the basis of knowledge about the distribution of parameter change together with the table data passed for the targets. At this stage the segmental assumption is dominant. However, before the real-time event and changeof-rate-of-change data are computed according to this framework and passed to the realtime data structure, the framework may be arbitrarily distorted by moving the nodes of the framework in both time and parameter value. The distortions may reflect the action of (a) local processes associated with the postures involved or with combinations of postures (for example, co-articulation); (b) global processes (associated for example, with vowel reduction or rate of utterance); or (c) processes using control information passed via the INT array or the console switches. Figure 1 illustrates the nature of the distortions.

Any process, controlled by any event, is capable, if required, of writing or re-writing data associated with all aspects of system operation, or of scheduling or cancelling the running of events that have yet to run. Modifications to the actions of other processes

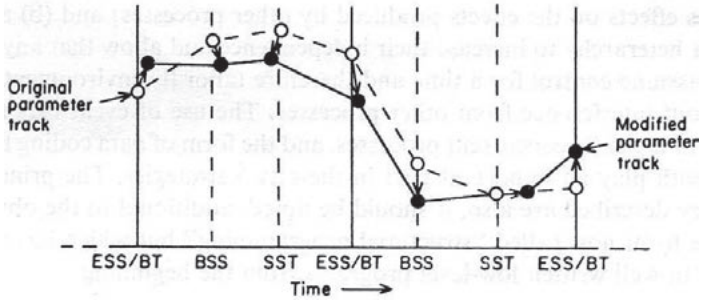

FIG. 1. Illustration of time framework distortions for one parameter

may also be produced by adding events to the real-time list that superimpose their effects on the basic pattern of parameter variation. Thus, since the parameter changes are controlled by specifying the rate-of-change in force at any given time, it is possible, by increasing the rate above the basic value for a fixed time and then decreasing it below its basic value by the same amount for the same time, to produce the effect of an added amount of (say) high-frequency noise, independently of how the noise parameter maybe varying, thus inserting a noise burst very simply. The old program structure made rather heavy weather of this kind of insertion. More than one effect may be superimposed in a similar fashion—a basic advantage of the rate-of-change run-length coding that is used to control the parameter values. Micro-intonationthat is, small deviations from the basic pitch contour due to segmental interaction with voicing frequency—is implemented using a similar strategy. The events that are inserted to the realtime list for such purposes are termed special parameter events, and they use data from a table based on a table address that forms part of the special parameter event code. Since there is very often a simple relation between the various positive and negative deviations required, the table needs fewer entries than might be supposed, bits in the event code being reserved to double and/or complement the change-of-rate-of-change value obtained from the table.

One rather surprising economy arose in connection with the pseudo- and real-time event lists themselves. Originally the program design called for circular queues without priority for the various ordered sets of data; and for priority queues for the two event lists, the latter being implemented as doubly linked lists, supposedly to allow fast, easy insertion of new events in their correct time order, regardless of when they were scheduled. Because new events are not added at random places in the event lists (the majority being added in the last few slots) it turns out to be faster to use a simple queue and merely dig in to the tail of the queue on the basis of an insertion sort than to use the more complicated structure. Furthermore, the space required for the handling routines and for the queues themselves is then roughly halved.

In summary, it may be said that the new program structure allows many different processes to run according to a program that is, in effect, partly determined by the processes themselves. This is one important dimension of flexibility. Secondly, the organization of the system is geared to allow the addition of new code with minimum effect on existing code. Thirdly, the interactions between the various parts of the potentially complex system are strictly controlled by two major strategies: (a) ensuring that any process that affects the output data produced by the system does so by linear super-

position of its effects on the effects produced by other processes; and (b) arranging the processes in a heterarchy to increase their independence and allow that any process can, in principle, assume control for a time and therefore tailor its environment and manage its data without interference from other processes. The use of event lists to permit the organization of logically concurrent processes, and the form of data coding for parameter generation, both play an important part in these two strategies. The principles of program structure described are also, it should be noted, additional to the obvious need to adhere to the form now called "structured programming" but which has been inherent and essential in well written low-level programs from the beginning.

Conclusion

It seems unlikely that the present system represents the ultimate basis for speech synthesis by machine. It is quite certain that we do not yet have the theoretical basis needed to judge such a matter. However, the flexibility of the scheme for synthesis, and the excellent mutual independence of the many processes with differing objectives that must be run for realistic approximations to real speech variation, have proved a welcome release from earlier problems. It is felt that we have, at last, a system within which we may try out a richer range of rules for synthesis at all levels, without so many constraining assumptions arising from mere programming difficulties, and which may be used to test some of the assumptions that led to its development. We have only just begun to explore the possibilities of the system, and to collect the data needed to improve our schematization of speech for synthesis by rules.

The author wishes to acknowledge with gratitude the support of the National Research Council of Canada for this work, under grant number A5261. His debt to colleagues at Essex University is considerable, especially to Ian Witten. Without his enthusiasm and critical interest the work would not have progressed as fast, and the joint Calgary-Essex project would have been stillborn.

References

BUCHANAN, B., SUTHERLAND, C. & FEIGENBAUM, E. A. (1969). HEURISTIC DENDRAL: a program for generating exploratory hypotheses in organic chemistry. In MELTZER, B. & MICHIE, D., Eds. *Machine Intelligence*, **4**, pp. 209-254. Edinburgh: Edinburgh University Press.

HILL, D. R. (1971). Man-machine interaction using speech. In ALT, F. L. & RUBINOFF, M., Eds, Yovrrrs, M., Guest Ed., *Advances in Computers*, **11**, pp. 165-230. New York: Academic Press.

HILL, D. R. (1972). An abbreviated guide to planning for speech interaction with machines: the state of the art. *International Journal of Man-Machine Studies*, **4** (4), 383-410.

HILL, D. R. & REID, N. A. (1977). An experiment on the perception of intonational features *International Journal Man-Machine Studies*, **9** (3), 337-347.

HOLMES, J. N. (1973). The influence of glottal waveform on the naturalness of speech from a parallel formant synthesiser. *IEEE Transactions on Audio and Electro-Acoustics*, **21** (3), 298305.

MACDOUGALL, M. H. (1970). Computer system simulation: an introduction. *Computing Surveys*, **2** (3), 191-209, September.

WITTEN, I. H. & SMITH, A. (1977). Synthesizing British English rhythm—a structured approach. *Proceedings of the 5th Man-Computer Communication Conference*, University of Calgary, Alberta, Canada, 26-27 May 1977, paper number 16.