Lecture #8: More About Regular Expressions
Key Concepts

Parsing

Definition: A ***parse tree*** for a regular expression ω over Σ is a rooted tree that can be defined as follows, for a given regular expression $\omega \in \Sigma^\star$:

(a) If ω is a symbol $\sigma \in \Sigma$ then the parse tree for ω has a single node, with label σ.

(b) If ω is the symbol "λ", then the parse tree for ω has a single node, with label "λ".

(c) If ω is the symbol "$\emptyset$", then the parse tree for ω has a single node, with label "$\emptyset$".

(d) If ω is the symbol "Σ", then the parse tree for ω has a single node, with label "Σ".

(e) If ω is a regular expression "$(\mu \cup \nu)$", for regular expressions μ and ν over Σ, then the parse tree for ω has a root with label '$\cup$" with two children. The first child is the root of a parse tree for μ, and the second child is the root of a parse tree for ν.

(f) If ω is a regular expression "$(\mu \circ \nu)$, for regular expressions μ and ν over Σ, then the parse tree for ω has a root with label "$\circ$" with two children. The first child is the root of a parse tree for μ, and the second child is the root of a parse tree for ν.

(g) If ω is a regular expression "$(\mu)^\star$, for a regular expression μ over Σ, then the parse tree for ω has a root with label "$\star$", with one child. The child is the root of a parse tree for μ.

An efficient algorithm for the computation of a parse tree, from a given regular expression over an alphabet Σ, is described in the lecture material.

Applications

Three ***applications*** of parse trees — involving useful computational problems concerning regular expressions — were described in the lecture material.

- The ***Recognition*** Problem: Given a string $R \in \Sigma^{\star}_{\text{regexp}}$, decide whether R is a regular expression over Σ.

  This problem can be solved, efficiently, by applying the above "parsing" algorithm, to try to construct a parse tree for R: R is a regular expression over Σ if and only if this attempt is successful.

- ***Conversion to NFA:*** Using a regular expression R, over an alphabet Σ, to produce a ***nondeterministic finite automaton*** with the same alphabet, Σ, and with the same language as R.

  A parse tree for the given regular expression can be used with a recursive algorithm to generate the desired nondeterministic finite automaton (using material from the constructive proof of the closure of the set of regular languages, under the regular operations, in the lecture material on "Regular Operations and Regular Expressions").

- The ***Membership*** Problem: Given a regular expression R over an alphabet Σ, and a string $\omega \in \Sigma^{\star}$, decide whether ω is in the language of R.

  This problem can be solved by constructing a nondeterministic finite automaton M (with alphabet Σ) with the same language as R, and then deciding whether $\omega \in L(M)$, using the process for this that was described for this in the lecture material on an "Introduction to Nondeterministic Finite Automata".

Regular Expressions in Practice

Regular operations used in software operations are not, quite, the same as the regular expressions defined in this course. Changes include the following:

- Changes needed to define "regular expressions" for alphabets Σ that include some, or all, of the special symbols "λ", "$\emptyset$", "(", ")", "$\cup$", "$\circ$", and "*".
- *Shortcuts* allowing "simpler" regular expression that do not include unnecessary brackets and support common cases — resulting in regular expressions that are shorter, but can also be harder to parse.

Industry standards for regular expressions, and software packages supporting their use, have been developed and used. Information concerning "regular expressions in practice" — which is ***for interest only*** in this course — is available in a supplemental document about this.