COMPILER DESIGN - REGULAR EXPRESSIONS

http://www.tutorialspoint.com/compiler_design/compiler_design_regular_expressions.htm

Copyright © tutorialspoint.com

The lexical analyzer needs to scan and identify only a finite set of valid string/token/lexeme that belong to the language in hand. It searches for the pattern defined by the language rules.

Regular expressions have the capability to express finite languages by defining a pattern for finite strings of symbols. The grammar defined by regular expressions is known as **regular grammar**. The language defined by regular grammar is known as **regular language**.

Regular expression is an important notation for specifying patterns. Each pattern matches a set of strings, so regular expressions serve as names for a set of strings. Programming language tokens can be described by regular languages. The specification of regular expressions is an example of a recursive definition. Regular languages are easy to understand and have efficient implementation.

There are a number of algebraic laws that are obeyed by regular expressions, which can be used to manipulate regular expressions into equivalent forms.

Operations

The various operations on languages are:

- Union of two languages L and M is written as

  L U M = {s | s is in L or s is in M}

- Concatenation of two languages L and M is written as

  LM = {st | s is in L and t is in M}

- The Kleene Closure of a language L is written as

  L* = Zero or more occurrence of language L.

Notations

If r and s are regular expressions denoting the languages Lr and Ls, then

- **Union** : $r|s$ is a regular expression denoting Lr U Ls
- **Concatenation** : rs is a regular expression denoting LrLs
- **Kleene closure** : r* is a regular expression denoting $L(r)$*
- r is a regular expression denoting Lr

Precedence and Associativity

- *, concatenation . , and | $pipesign$ are left associative
- * has the highest precedence
- Concatenation . has the second highest precedence.
- | $pipesign$ has the lowest precedence of all.

Representing valid tokens of a language in regular expression

If x is a regular expression, then:

- x* means zero or more occurrence of x.

  i.e., it can generate { e, x, xx, xxx, xxxx, … }

- x+ means one or more occurrence of x.

  i.e., it can generate { x, xx, xxx, xxxx … } or x.x*

- x? means at most one occurrence of x

  i.e., it can generate either {x} or {e}.

  [a-z] is all lower-case alphabets of English language.

  [A-Z] is all upper-case alphabets of English language.

  [0-9] is all natural digits used in mathematics.

Representing occurrence of symbols using regular expressions

letter = [a – z] or [A – Z]

digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 or [0-9]

sign = [+ | -]

Representing language tokens using regular expressions

Decimal = $sign^{?}digit^{+}$

Identifier = $letterletter|digit^{*}$

The only problem left with the lexical analyzer is how to verify the validity of a regular expression used in specifying the patterns of keywords of a language. A well-accepted solution is to use finite automata for verification.

Loading [MathJax]/jax/output/HTML-CSS/jax.js