Compiler Design
Lexical Analysis
Specification of Tokens

conf. dr. ing. Ciprian-Bogdan Chirila

chirila@cs.upt.ro

http://www.cs.upt.ro/~chirila

Outline

- Strings and Languages
- Operations on Languages
- Regular Expressions
- Regular Definitions
- Extensions of Regular Expressions

Regular Expressions

- are important for specifying lexeme patterns
- cannot express all possible patterns
- they are effective in specifying those types of patterns that we actually need for tokens
- we will study their formal notation
- will be used in lexical analyzer generators

Strings and Languages

- Alphabet – any finite set of symbols
    - example of symbols: letters, digits, punctuation
  - {0, 1} – binary alphabet
  - ASCII – important alphabet used in many software systems
  - Unicode – ~100,000 characters alphabet

Strings and Languages

String (over an alphabet) – finite sequence of symbols drawn from that alphabet

- a.k.a. "*sentence*" or "*word*" in language theory
- $|s|$ – length of string s
  - "banana" has the length 6
- ε – empty string
  - has the length 0

Strings and Languages

- Language – any countable set of strings over some fixed alphabet
  - e.g.:
    - Abstract languages: Ø, empty set, or {ε}
    - All syntactically well-forms C programs
    - The set of all grammatically correct English sentences
  - Does not require that any meaning be ascribed to the strings in the language

Terms for Parts of Strings

- Prefix of string *s*
  - Any string obtained by removing zero or more symbols from the end of *s*
    - E.g.: ban, banana, and ε are prefixes of banana
- Suffix of string *s*
  - Any string obtained by removing zero or more symbols from the beginning of *s*
    - E.g.: nana, banana, and ε are suffixes of banana

Terms for Parts of Strings

- Substring of string *s*
  - Any string obtained by deleting any prefix and any suffix from *s*
    - E.g.: banana, nan, and ε are substrings of banana
- Proper prefixes, suffixes, and substrings of string *s*
  - Those prefixes, suffixes, and substrings of s that are
    - not ε or
    - not equal to s itself
- Subsequence of *s*
  - Any string formed by deleting zero or more not necessarily consecutive positions of s
    - E.g.: baan is a subsequence of banana

Terms for Parts of Strings

- Concatenation of strings *x* and *y*
  - String formed by appending *y* to *x*, denoted *xy*
  - *x* = dog, *y* = house => *xy* = doghouse
  - Empty string is the identity under concatenation
    - $\varepsilon s = s\varepsilon = s$
- Exponentiation based on concatenation as a product
    - $s^0 = \varepsilon$
    - for all $i>0$ we define s^i to be $s^{i-1}s$
    - $s^1 = s$
    - $s^2 = ss$
    - $s^3 = sss$

Operations on Languages

- Union
  - The familiar operation on sets
- Concatenation
  - All strings formed by taking a string from the first language and a string from the second language, in all possible ways, and concatenating them
- (Kleene) closure L^* of a language L
  - The set of strings formed by concatenating L zero or more times
  - $L^0 = \{\varepsilon\}$
  - L^+ is the positive closure is the same as the Kleene closure, but without the term L^0

Operations on Languages

OPERATION	DEFINITION AND NOTATION
Union of L and M	$L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$
Concatenation of L and M	$LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$
Kleene closure of L	$L^* = \cup_{i=0}^{\infty} L^i$
Positive closure of L	$L^+ = \cup_{i=1}^{\infty} L^i$

Examples

- L={A,B,... , Z, a,b,..., z}
- D={0,1,... 9}
  - $L \cup D$ is the set of letters and digits with 62 strings of length one, either one letter or one digit;
  - LD is the set of 520 strings of length two, each consisting of one letter followed by one digit;
  - L^4 is the set of all 4-letter strings;
  - L^* is the set of all strings of letters, including ε;
  - $L(L \cup D)^*$is the set of all strings of letters and digits beginning with a letter;
  - D^+ is the set of all strings of one or more digits;

Regular Expressions

- How can we describe the set of valid C identifiers?
- Regular expressions - describing all the languages that can be built from union, concatenation, and closure operators applied to the symbols of some alphabet

Regular Expressions

- if *letter_* is established to stand for any letter or the underscore
- *digit* is established to stand for any digit
- then we could describe the language of C identifiers by:

  *letter_ (letter_| digit)**

- the vertical bar above means union
- the parentheses are used to group subexpressions
- the star means "zero or more occurrences of"
- the juxtaposition of *letter*, with the remainder of the expression signifies concatenation

Regular Expressions

- The regular expressions are built recursively out of smaller regular ones
- Each regular expression r denotes a language *L(r)*, which is also defined recursively from the languages denoted by r's subexpressions
- BASIS: There are two rules that form the basis
  - 1. ε is a regular expression, and *L(ε)* is {ε}, that is, the language whose sole member is the empty string.
  - 2. If *a* is a symbol in Σ, then **a** is a regular expression, and L(**a**) = {*a*}, that is, the language with one string, of length one, with *a* in its one position.
    - Note that by convention, we use italics for symbols, and boldface for their corresponding regular expression

Regular Expressions

- INDUCTION: There are four parts to the induction whereby larger regular expressions are built from smaller ones.
- Suppose r and s are regular expressions denoting languages *L(r)* and *L(s)*, respectively.
  - 1. (r)|(s) is a regular expression denoting the language *L(r)* U *L(s)*.
  - 2. (r)(s) is a regular expression denoting the language *L(r)L(s)*.
  - *3.* (r)* is a regular expression denoting (L(r))*.
  - 4. (r) is a regular expression denoting *L(r)*. This last rule says that we can add additional pairs of parentheses around expressions without changing the language they denote.

Regular Expressions

- Regular expressions often contain unnecessary pairs of parentheses
- We may drop certain pairs of parentheses if we adopt the conventions that:
  - a) The unary operator * has highest precedence and is left associative
  - b) Concatenation has second highest precedence and is left associative
  - c) | has lowest precedence and is left associative
- for example, we may replace the regular expression (a)|((b)*(c)) by a|b*c

Algebraic laws for regular expressions

LAW	DESCRIPTION
$r\|s = s\|r$	$\|$ is commutative
$r\|(s\|t) = (r\|s)\|t$	$\|$ is associative
$r(st) = (rs)t$	Concatenation is associative
$r(s\|t) = rs\|rt;\ (s\|t)r = sr\|tr$	Concatenation distributes over $\|$
$\epsilon r = r\epsilon = r$	ϵ is the identity for concatenation
$r^* = (r\|\epsilon)^*$	ϵ is guaranteed in a closure
$r^{**} = r^*$	* is idempotent

Regular Definitions

- For notational convenience, we may wish to give names to certain regular expressions and use those names in subsequent expressions, as if the names were themselves symbols.
- If Σ is an alphabet of basic symbols, then a *regular definition* is a sequence of definitions of the form:

$$\begin{gathered} \mathrm{d}_1 \rightarrow \mathrm{r}_1 \\ \mathrm{d}_2 \rightarrow \mathrm{r}_2 \\ \cdots \\ \mathrm{d}_\mathrm{n} \rightarrow \mathrm{r}_\mathrm{n} \end{gathered}$$

where:

1. Each d_i is a new symbol, not in Σ and not the same as any other of the d's, and
2. Each r_i is a regular expression over the alphabet $\Sigma \cup \{d_1, d_2, \dots, d_{i-1}\}$

Example – C identifiers

- C identifiers are strings of
  - letters
  - digits
  - underscores

$$
\begin{array}{rcl}
\mathit{letter\_} & \rightarrow & \mathtt{A} \mid \mathtt{B} \mid \cdots \mid \mathtt{Z} \mid \mathtt{a} \mid \mathtt{b} \mid \cdots \mid \mathtt{z} \mid \mathtt{\_} \\
\mathit{digit} & \rightarrow & \mathtt{0} \mid \mathtt{1} \mid \cdots \mid \mathtt{9} \\
\mathit{id} & \rightarrow & \mathit{letter\_} \; (\; \mathit{letter\_} \mid \mathit{digit} \;)^*
\end{array}
$$

Example – unsigned numbers

$$
\begin{array}{rcl}
\mathit{digit} & \rightarrow & \mathtt{0} \mid \mathtt{1} \mid \cdots \mid \mathtt{9} \\
\mathit{digits} & \rightarrow & \mathit{digit}\ \mathit{digit}^{*} \\
\mathit{optionalFraction} & \rightarrow & \mathtt{.}\ \mathit{digits} \mid \epsilon \\
\mathit{optionalExponent} & \rightarrow & (\ \mathtt{E}\ (\ \mathtt{+} \mid \mathtt{-} \mid \epsilon\ )\ \mathit{digits}\ ) \mid \epsilon \\
\mathit{number} & \rightarrow & \mathit{digits}\ \mathit{optionalFraction}\ \mathit{optionalExponent}
\end{array}
$$

Extensions of Regular Expressions

- One or more instances

- Zero or one instance

- Character classes

One or more instances

- The unary, postfix operator + represents the positive closure of a regular expression and its language
- That is, if r is a regular expression, then
  - $(r)^+$ denotes the language $(L(r))$.
  - The operator $^+$ has the same precedence and associativity as the operator *

Zero or one instance

- The unary postfix operator ? means "zero or one occurrence"
- That is, r? is equivalent to r|ε, or
- put another way, $L(r?) = L(r) \cup \{e\}$.
- The ? operator has the same precedence and associativity as * and +

Character classes

- A regular expression $a_1|a_2| \ldots |a_n$
- where the a_i's are each symbols of the alphabet, can be replaced by the shorthand $[a_1a_2\ldots a_n]$
- More importantly, when $a_1,a_2,\ldots,a_n$ form a logical sequence
  - e.g. consecutive uppercase letters, lowercase letters, or digits,
- we can replace them by a_1-a_n , that is, just the first and last separated by a hyphen
- [abc] is shorthand for a|b|c
- [a-z] is shorthand for a|b|…|z

Bibliography

- Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman – Compilers, Principles, Tehcniques and Tools, Second Edition, 2007