CST8152- Compilers
Article #7
Language specification

Grammars

Noam Chomsky's Grammar Hierarchy

1. Regular Grammars
2. Context Free Grammars (CFG or BNF)
3. Context Sensitive Grammars
4. General

A Context Free Grammar (CFG) is defined by the following four components:

1. A finite set of terminal symbols (***terminals***) or a final terminal vocabulary $\mathbf{V_t}$. For the lexical grammar the terminals are the alphabet; for the syntactic grammar the terminals are the token set produced by the scanner and defined by the lexical grammar.
2. A finite set of ***nonterminals*** or a nonterminal vocabulary $\mathbf{V_n}$ .Nonterminals are not part of the language. They are intermediate symbols used to define the grammar for the language.
3. A finite set of ***productions*** (rewriting or replacement or substitution or derivation rules) **P**. Productions have the form:

   $A \text{ -> } X_1X_2X_3\ldots X_m$
   where $A \in V_{n,} X_i \in V_n \cup V_t$, $1 \le i \le m, m > 0$
   and
   $A \text{ -> } \varepsilon$ (empty) (m = 0) is a valid production

4. A **start** (or **goal**) symbol **S**. The start symbol $S \in V_n$ (S belong to V_n) is always the root of the parse tree.

Following the definition above, a CFG is the four-toupe $\mathbf{G = (V_t, V_n, P, S)}$.
L(G) is the language defined or generated by the grammar.

Kleene's Theorem

In 1956 Stephen Cole Kleene formulated and proved the following theorem:

The language that can be defined by

1. Regular Expressions (or Regular Grammar)

or

2. Transition graph (transition or state diagrams)

or

3. Finite Automaton (Finite State Machine)

can be defined by all three methods.

Regular Expressions

Regular expressions are a convenient notation (or means or tools) for specifying certain simple (though possibly infinite) set of strings over some alphabet. As such, they can be used for describing lexical symbols or for specifying the structure (pattern) of the token used in a programming language. A regular expression can be used to construct a Deterministic Finite Automaton (DFA) which therefore can recognize strings (words) of the grammar, which is the purpose of the Scanner.

A regular expression is a shorthand equivalent to a regular grammar; it is a pattern that strings must match or conform to, to be valid words (or sentences).
The sets of strings defined by regular expression are termed regular sets.

As defined above, the grammar is the following tuple: **G = (V_t, V_n, P, S).** If we denote **r** as a regular expression, then since **G** and **r** are equivalent, they must produce the same languages, so:

$$L(G) = L(r)$$

To define the strings, regular expressions (as any expression notation) use operands and operations. The operands are alphabet symbols or strings defined by regular expressions (regular definitios). The standard operations are catenation (concatenation) , union or alternation (|), and Kleene closure (*)
Regular expressions use the metasymbols **|**, **(**, **)**, **{**, **}** , **[**, **]**, ***** , **+** (and others **?**, **^**)
to define its operations.

Alphabet

An alphabet ∑ is a finite, non-empty, set of symbols.
For example:

- the binary alphabet is {0, 1}
- the decimal alphabet is {0,1,2,3,4,5,6,7,8,9}

(note the metasymbols { , and } used here that are not in the alphabet)

For the scanner, the alphabet may be characters in the ASCII character set. For the parser the alphabet is the set of tokens produced by the scanner. The set of keywords is also an alphabet (even though by itself it is insufficient to form a language):

{if, else, for, while, switch, case …}

String

A string is a finite set of symbols from an alphabet (not necessarily in a grammar).
For example:
For the alphabet ∑ = {a, b, c}
the set of all possible strings is

{a, b, c, ab, ac, bc, ba, ca, cb, abc, acb, bac, bca, cab, cba}

The order of symbols in a string matters.

Empty string

ε is the **empty string.** It is the string consisting of no symbols.

The **length of a string** s is |s| and is equal to the number of symbols in the string.
So: |ε| = 0, |abc| = 3.

Single Character Regular Expressions

a. A single character.

A regular expression can be a pattern for a single character from the alphabet . This is the simplest case. Consider an alphabet

∑ = {a, b, c, d} .

If we write the regular expression `b`, then:

L(b) = {b}

In plain language, the regular expression `b` means simply the set of characters consisting only of `b`, which means that the regular expression `b` is the character `b`. On the same token, a string s is regular expression denoting a set containing only s. if it contains meta-characters, s can ne quoted to avoid ambiguity (“s”).

b. The empty string

A regular expression can be the empty string ε. This is defined by:

$$L(\varepsilon) = \{\varepsilon\}$$

The regular expression ε means the set containing only the empty string.

c. The regular expression that matches nothing Φ.

Note this is not the same as the empty string. Defined by:

$$L(\Phi) = \{\ \}$$

Φ is the pattern for nothing; it generates the set containing nothing.

By contrast, ε is the pattern for the set that contains the string that contains no characters ☺

Concatenation of strings

Concatenation joins the strings together.
For example:
If x and y are strings, then their concatenation is the string xy.
For the alphabet {a, b, c}, if x = ab and y = bc, then xy = abbc

Concatenation with the empty string ε leaves a string unchanged:

εx = xε = x

Operations with Regular Expressions

a. Alternatives – the | operator

If **x** and **y** are regular expressions, then **x | y** is the regular expression with alternatives. It means a string that matches either the regular expression **x** or the regular expression **y**. Formally:

L(x | y) = L(x) L(y)

For example if ∑ = {a, b, c, d} then L(a|b) = {a,b} and L(c|d) = {c,d} so
a|b is either a or b
c|d is either c or d

Like wise, the regular expression a| ε is either a or ε

b. Concatenation (writing adjacent letters or strings, or . or ,)

The concatenation of two regular expressions x and y is xy. It means take one of x's string followed by one of y's strings. Formally:

L(xy) = L(x)L(y)

More complex expressions can be written.

Example 1 **(a|b)c**
The parenthesis enforces alternation before concatenation.
L((a|b)c) = L(a|b)L(c) = {a,b} {c} = {ac, bc}

Example 2 **(a|b)(a|b)**
The parenthesis enforces alternation before concatenation.
L((a|b)(a|b) = L(a|b)L(a|b) = {a,b} {a,b} = {aa, ba, ab, bb}

Example 2 **aa|ba|ab|bb**
This is just another way of writing example 2

Kleene Closure

For a regular expression r, Kleene Closure is defined by:

L(r*) = L(r)*

which means concatenation with all powers of L. As an example, from our alphabet,
L((a|bb)*) = L(a|bb)* = $L(a|bb)^0 + L(a|bb)^1 + L(a|bb)^2 + L(a|bb)^3$ +…
and
$L(a|bb)^0$ = {ε}
$L(a|bb)^1$ = {a,bb}
$L(a|bb)^2$ = {a,bb}{a,bb} = {aa, abb, bba, bbbb}
$L(a|bb)^3$ = {aa, abb, bba, bbbb}{a,bb} = {aaa,abba,bbaa,bbbba,aabb,abbbb,bbabb,bbbbbb}
so
L((a|bb)*) = {ε, a, bb, aa, abb, bba, bbbb, aaa, abba, bbaa, bbbba, aabb, abbbb, bbabb, bbbbbb,…}

Nonstandard Regular Expressions operations

Positive Closure (One or more +)

$a^+ = aa^*$ (also $a^* = a^+ | \varepsilon$)

Exponentiation or Power operation (exactly k)

$a^k = aaa\ldots a$ **(exactly k times)**

Optional Inclusion (Zero or one ?)

$a? = \varepsilon \mid a$

Character classes
Specify a range of characters or numbers that follow a sequence.

[a-z] means any character in the range a to z.
[A-Z] means any character in the range A to Z.
A regular expression for the pattern for an identifier that begins with a letter or an underscore and is followed by any number of numbers and letters is [a-zA-Z_][A-Za-z0-9]*.

A complement character class is specified using the ^ or (~) symbols, or **Not** operator

[^a-z] matches any character except a to z.

Comment = // (^CR)*CR

Precedence of Operators

The parentheses, ( ) allow us to override the precedence of operators:
In order of decreasing precedence:

( )	- grouping
[ ]	- character classes
*** , +, k, ?**	- (Kleene star), positive closure, power, opt. inclusion
, . , , ,	(or writing adjacent) - concatenation
|	- the alternation operator

Formal Definition of Regular Expresiions
Each regular expression denotes (defines) a set of strings (regular sets). Formally, regular expressions are defined as follows.

- Φ is a regular expression denoting the empty set.
- ε is a regular expression denoting the set that contains only the empty string.
- A string s is a regular expression denoting a set containing only s.
- If A and B are regular expressions, then A | B, AB, and A* are also regular expressions. In this case, A and B are sometimes called regular definitions

Regular expressions define sets of strings. Regular expression operations are operations on sest of strings.

Operations on Sets of Strings

1. Concatenation of sets

The concatenation of two sets A and B is defined by:
AB = { xy | x in A and y in B }
which reads “the set of strings xy such that x is in A and y is in B”.
For example. If A = {a,b} and B = {c,d}
then AB = { ac, ad, bc, bd }

2. Powers of sets

The power of a set A:
A^4 = { x | 4-symbol string }
which reads “the set of strings with four symbols”.
This is just repeated concatenation:
A^0 = { ε }, A^1 = A, A^2 = AA, A^3 = AAA, …
(note that A^0 = { ε } for any set)
For example. If A = {a,b} then
A^0 = { ε }, A^1 = { a,b }, A^2 = { aa, ab,ba, ,bb },
A^3 = { aaa, aab,aba, abb, baa, bab, bda, bbb }

3. Union of sets

The union of two sets A and B is defined by:
AUB = { x | x in A or x in B }
which reads “the set of strings x such that x is in A or x is in B”.
For example. If A = {a,b} and B = {c,d}
then AUB = { a, b, c, d }

4. Kleene Closure

The Kleene closure of a set A is the * operator defined as the set of all strings including the empty string:

$$A^* = \bigcup_{i=0}^{\infty} A^i$$

It is the union of all powers of A.
$A^* = A^0 + A^1 + A^2 + A^3 + \ldots$
For example, if A = {a,b} then:
A* = { ε } + { a,b } + { ab,ba,aa,bb }
+ { aaa, aab,aba, abb, baa, bab, bda, bbb } +…

= { ε, a, b, ab, ba, aa, bb, aaa, aab,aba, abb, baa, bab, bda, bbb, …}

5. **Positive Closure**

The positive closure is the + operator defined as the set of all strings excluding the empty string:

$$A* = \bigcup_{i=1}^{\infty} A^i$$

It is the union of all powers of A except the empty string.

A+ = $A^1 + A^2 + A^3 + \ldots$

For example, if A = {a,b} then:

A+ = { a,b } + { ab,ba,aa,bb } + { aaa, aab,aba, abb, baa, bab, bda, bbb } +…

= { a, b, ab, ba, aa, bb, aaa, aab,aba, abb, baa, bab, bda, bbb, …}

Kleene closure is the union of the set containing the empty string and positive closure:

$A^* = A^+$ U A0

Examples

Let L be the set {A, B, … Z, a, b, … z}
Let D be the set {0, 1, …….9}

1. L U D is the set of letters and digits
   {A, B, … Z, a, b, … z, 0, 1, …….9}
2. LD is the set of strings consisting of a letter followed by a digit
   {A0, A1, A2 .., B0, B1,… Z9, a0, b0, a1, b1 … }
3. L^4 is the set of all four-letter strings
4. L* is the set of all strings of letters, including ε
5. L(LUD)* is the set of all strings of letters and digits beginning with a letter
6. D^+ is the set of all strings of one or more digits

Limitations of regular expressions

Regular expressions are simpler than grammars but are less powerful. Some patterns can only be derived from grammars (matched parenthesis, repeating patterns).

Every pattern that can be described by a regular expression can also be described by a grammar.

Example

Regular Expression: (a|b)*abb

a or b, repeated zero or more times, followed by abb:

{abb, aabb, aaabb, …, babb, bbabb, bbbabb…}

Equivalent grammar:

$A_0 \rightarrow aA_0|bA_0|aA_1$

$A_1 \rightarrow bA_2$

$A_2 \rightarrow bA_3$

$A_3 \rightarrow \varepsilon$