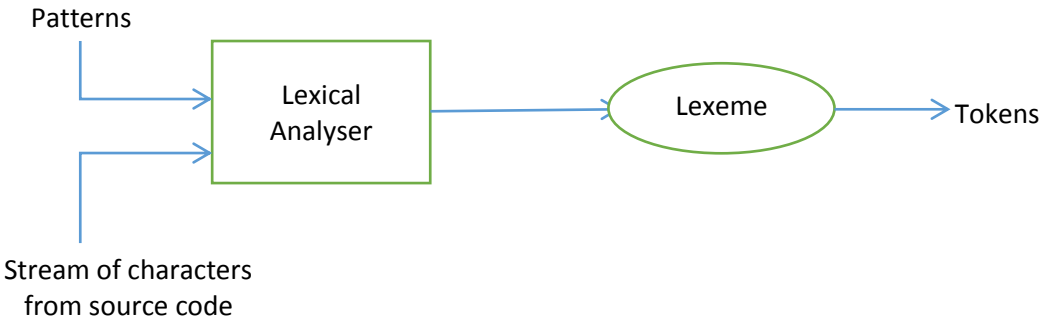
Lecture – 8 (6/8/13)

As in the last lecture, we were discussing about the lexical analyser and how we need to identify the different forms of patterns pertaining to particular language syntax. These patterns help us in the generation of lexemes and then tokens. To identify or to study these patterns, we use the concepts of formal language or more specifically regular expressions.



Before learning regular expressions, we need to know some definitions which are used to define regular expressions.

Alphabet: Set of Symbols (digits, letters, special characters, etc.)
String: Defined on alphabet (Countable Infinite Set)
Language: Countable set of Strings (Finite/ Infinite)

Regular Expression:

Regular Expressions consist of constants and operator symbols that denote sets of strings and operations over these sets, respectively. Given a finite alphabet ∑, the following constants are defined as regular expressions:

- **Empty set:** Φ denoting the set Φ .
- **Empty string:** ε denoting the set containing only the "empty" string having no characters at all.
- **Literal:** a in ∑ denoting the set containing only the character a.

Let two languages (regular) be defined over various alphabets as:

L = {a,b,……,z} D = {0,1,….,9}

Then the following operations are the valid regular expressions:

1. **Union:** L U D = { S | S ϵ L or S ϵ D}
2. **Concatenation:** LD = {ST | S ϵ L , T ϵ D}
3. **k-closure:** $L^* = L^0UL^1U……L^\infty$
4. **p-closure:** $L^+ = L^1UL^2U……L^\infty$

example:

L(LUD)* = x, x2, y2a, ……
(in this language we cannot use strings starting with integers.)

Therefore, by this example we can say:

Language ⇔ Pattern

Operations on regular languages:

1. ϵ => $L(\epsilon)$
2. $a \epsilon \sum$ => $L(a)$
3. let r and s be two regular languages, $(r + s)$ => $L(r) \cup L(s)$
4. $rs = L(r).L(s)$
5. $r^* = L(r)^*$

Therefore, using these basic operations, we can define more and more complex languages.

In the coming section we will apply all the knowledge we have gained so far to construct some regular expressions that are related to C language:

1. Identifier names:

   As the identifier names should not start with any number or should not contain any special character(except "underscore"), therefore the following expression will define the language for identifier names;

   Let $\sum^d$ (set of positive integers) = {0,1,2,…,9} $\sum^l$ (set of English alphabets) = {a,b,…..,z}
   and $\sum^u$ ("underscore") = { _ }
   then the language for identifier will be:

   $$L = (\sum^l \cup \sum^u).(\sum^l + \sum^u + \sum^d)$$

2. Unsigned numbers:

   Let two sets be digits = {0,1,2,….,9} => digits = $digit^+$
   Therefore decimal numbers can be defined as:
   decimal = .digits | ϵ

   Now, floating point numbers can be defined as:
   float = digits.decimal (concatenation)

   and similarly we can define signed or exponent numbers.

Therefore, all these discussions prove the importance of patterns and how to detect those patterns in Lexical Analysis stage of Compilation. Later, we will see how to actually compare a string of characters with different patterns using various DFAs and how to construct them using the given regular expressions.