Lexical Analysis II (Scanning)

Course mentor:
Dr. Nouf Altamami
Dr. Nayyar Ahmed Khan
+966-593324667
nayyar@su.edu.sa

Agenda

- Finite automata (skip NFAs)
- From regular expressions to DFAs (skip NFAs)
- Implementation of a scanner
- Using LEX (will be covered in a lab session)

Lexical Analysis

- Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.

- If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer when it demands.

Tokens

Lexemes are said to be a sequence of characters (alphanumeric) in a token. There are some predefined rules for every lexeme to be identified as a valid token. These rules are defined by grammar rules, by means of a pattern. A pattern explains what can be a token, and these patterns are defined by means of regular expressions.

In programming language, keywords, constants, identifiers, strings, numbers, operators and punctuations symbols can be considered as tokens.

For example

- int value = 100;
- int (keyword), value (identifier), = (operator), 100 (constant) and ; (symbol).

Specifications of Tokens

Alphabets

Strings

Special Symbols

Language

Regular Expressions

- 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**.

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 L(r) and L(s), then
- **Union** : (r)|(s) is a regular expression denoting L(r) U L(s)
- **Concatenation** : (r)(s) is a regular expression denoting L(r)L(s)
- **Kleene closure** : (r)* is a regular expression denoting (L(r))*
- (r) is a regular expression denoting L(r)

Precedence and Associativity

*, concatenation (.), and | (pipe sign) are left associative

* has the highest precedence

Concatenation (.) has the second highest precedence.

| (pipe sign) has the lowest precedence of all.

Representing valid tokens of a language in regular expression

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 = [+ | -]

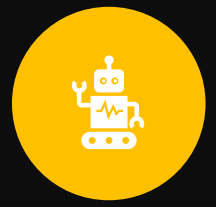
Finite Automata

- Finite automata is a state machine that takes a string of symbols as input and changes its state accordingly. Finite automata is a recognizer for regular expressions. When a regular expression string is fed into finite automata, it changes its state for each literal. If the input string is successfully processed and the automata reaches its final state, it is accepted, i.e., the string just fed was said to be a valid token of the language in hand.

The transition function (δ) maps the finite set of state (Q) to a finite set of input symbols (Σ),
$Q \times \Sigma \rightarrow Q$

Finite set of states (Q)

Finite set of input symbols (Σ)

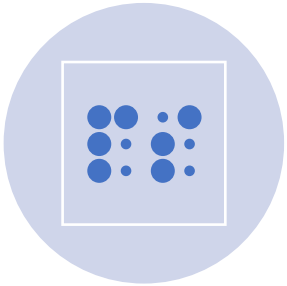


One Start state (q0)

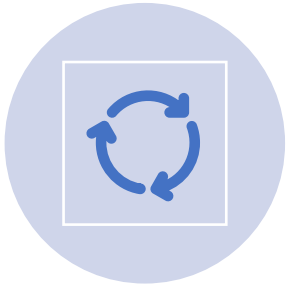
Set of final states (qf)

Transition function (δ)

Finite Automata Construction

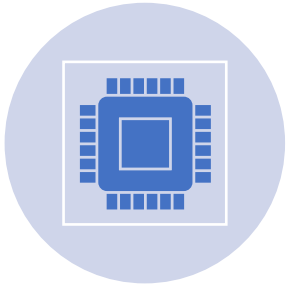


States : States of FA are represented by circles. State names are written inside circles.



Start state : The state from where the automata starts, is known as the start state. Start state has an arrow pointed towards it.

Intermediate states : All intermediate states have at least two arrows; one pointing to and another pointing out from them.



Final state : If the input string is successfully parsed, the automata is expected to be in this state. Final state is represented by double circles.

Example : We assume FA accepts any three-digit binary value ending in digit 1. FA = $\{Q(q_0, q_f), \Sigma(0,1), q_0, q_f, \delta\}$