Introduction to Theory of Computation

Lecturer:- Dr.**Najat Hameed**

Computer Science Department

University of Basrah

Reference:- Introduction to Computational Theory by Daniel I.A. Cohen.

Background

- Theory of computation is the theoretical study of capabilities and limitations of computers.
- Understanding of basic concepts in the theory of computation through simple models of computational devices
- This theory is very much relevant to practice, for example, in the design of new programming languages, compilers, string searching, pattern matching, computer security, artificial intelligence, etc

Mathematical preliminaries

- Throughout this course, we will assume that you know the following mathematical concepts:

1. A set is a collection of well-defined objects.
2. The set of natural numbers is N = {1, 2, 3, . . .}.
3. The set of integers is Z = {. . . ,−3,−2,−1, 0, 1, 2, 3, . . .}.
4. The set of rational numbers is Q = {m/n : m ∈ Z, n ∈ Z, n 6= 0}.
5. The set of real numbers is denoted by R.

Mathematical preliminaries

6. If A and B are sets, then A is a subset of B, written as $A \subseteq B$, if every element of A is also an element of B. For example, the set of even natural numbers is a subset of the set of all natural numbers. Every set A is a subset of itself, i.e., $A \subseteq A$. The empty set is a subset of every set A, i.e., $\emptyset \subseteq A$.

7. If B is a set, then the power set P(B) of B is defined to be the set of all subsets of B:

$$P(B) = \{A : A \subseteq B\}.$$

Observe that $\emptyset \in P(B)$ and $B \in P(B)$.

Mathematical preliminaries

8. If A and B are two sets, then

(a) Their union is defined as:

$A \cup B = \{x : x \in A \text{ or } x \in B\}$,

(b) Their intersection is defined as

$A \cap B = \{x : x \in A \text{ and } x \in B\}$,

(c) Their difference is defined as

$A \setminus B = \{x : x \in A \text{ and } x\ 6\!\in B\}$,

(d) The Cartesian product of A and B is defined as

$A \times B = \{(x, y) : x \in A \text{ and } y \in B\}$,

Mathematical preliminaries

(e) The complement of A is defined as A = {x : x 6∈ A}.

9. A binary relation on two sets A and B is a subset of A × B.

10. A function f from A to B, denoted by f : A → B, is a binary relation R, having the property that for each element a ∈ A, there is exactly one ordered pair in R, whose first component is a. We will also say that f(a) = b or f maps a to b, or the image of a under f is b. The set A is called the domain of f, and the set {b ∈ B : there is an a ∈ A with f(a) = b}is called the range of f.

Basic Definitions

1. **Alphabet** - a finite set of symbols.
   - Notation: Σ .
   - Examples: Binary alphabet {0,1},
     English alphabet {a,...,z,!,?,...}
2. **String over an alphabet** Σ - a finite sequence of symbols from Σ.
   - Notation: (a) Letters u, v, w, x, y, and z denote strings.
     (b) Convention: concatenate the symbols. No parentheses or commas used.
   - Examples: 0000 is a string over the binary alphabet.
     a!? is a string over the English alphabet.

Basic Definitions

3. **Empty string:** e or ε denotes the empty sequence of symbols.
4. **The length of a string w**, denoted by |w|, is the number of symbols contained in w. The empty string denoted by ε, is the string having length zero. Thus, |00100| = 5, |aab| = 3, | ε | = 0. For example, if the alphabet is equal to {0, 1}, then, 10, 1000, 0,101, and ε are strings over Σ , having lengths 2, 4, 1, 3, and 0, respectively.

Basic Definitions

5. **Language over alphabet Σ** - a set of strings over Σ.
   - Notation: L.
   - Examples:

     {0, 00, 000, ...} is an "infinite" language over the binary alphabet.

     {a, b, c} is a "finite" language over the English alphabet.

Basic Definitions

6. **Empty language** - empty set of strings.
   Notation: Φ.

- **Binary operation on strings**: Concatenation of two strings u.v - concatenate the symbols of u and v.
  - Notation: uv
  - Examples:
    - 00.11 = 0011.
    - ε.u = u.ε = u for every u. (identity for concatenation)

Binary relations on strings

1. **Prefix** - u is a prefix of v if there is a w such that v = uw.
   - Examples:
     - ε is a prefix of 0 since 0 = ε0
     - apple is a prefix of appleton since appleton = apple.ton
2. **Suffix** - u is a suffix of v if there is a w such that v = wu.
   - Examples:
     - 0 is a suffix of 0 since 0 = ?
     - ton is a suffix of appleton since ?

Binary relations on strings

3. **Substring** - u is a substring of v if there are x and y such that v = xuy.
   - Examples:
     - let is a substring of appleton since appleton = app.let.on
     - 0 is a substring of 0 since 0 = epsilon.0.epsilon

Observe that prefix and suffix are special cases of substring.

Relevance of strings and languages

- Each language in the linguistic field consists of three entities; letters, words, and sentences.
- Set of characters shape word, group of words collect sentences which form paragraph and etc.
- Not every set of letters can shape valid words and not every collection of words can make up a valid sentence.

Relevance of strings and languages

- Similarly, in computer languages, a certain set of characters form a word (e.g. while, for, and so on). Certain collection of words shape commands (e.g. for (i > 0; i < 100; i++), and certain set of commands compose program.
- **Theory of Formal Languages** is an interesting set of string of symbols that obey a set computer language rules. The set of symbols does not focus on the meaning. Formal language theory concentrates on syntax not on semantics (i.e. focuses on word spelling, not on the word meaning).

Relevance of strings and languages

- **Closure (*) of the alphabet**, is a language that contains a set of finite length of strings (including ε). Each string is shaped from concatenation of the alphabet elements. Closure (*) sometimes is called Kleene star.
- *L is a said to be a language over alphabet Σ, only if $L \subseteq \Sigma^*$*

  ➔ this is because Σ^* is the set of all strings (of all possible length including 0) over the given alphabet Σ

Relevance of strings and languages

Examples:

1. Let L be *the* language of all strings consisting of *n* 0's followed by *n* 1's:

   L = {ε, 01, 0011, 000111,…}

2. Let L be *the* language of all strings of with equal number of 0's and 1's:

   L = {ε, 01, 10, 0011, 1100, 0101, 1010, 1001,…}

Definition: **Ø denotes the Empty language**

Kleene Closure (*)

Let Σ be an alphabet.

- Σ^k = the set of all strings of length k
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$

Kleene Closure

- Kleene Closure of a given language L:
  - $L^0 = \{\varepsilon\}$
  - $L^1 = \{w \mid \text{for some } w \in L\}$
  - $L^2 = \{ w_1w_2 \mid w_1 \in L, w_2 \in L \text{ (duplicates allowed)}\}$
  - $L^i = \{ w_1w_2\ldots w_i \mid \text{all } w\text{'s chosen are} \in L \text{ (duplicates allowed)}\}$
  - (Note: the choice of each w_i is independent)
  - $\mathbf{L^*} = \bigcup_{i\geq 0} L^i$ (arbitrary number of concatenations)

Example:

- Let $L = \{ 1, 00\}$
  - $L^0 = \{\varepsilon\}$
  - $L^1 = \{1,00\}$
  - $L^2 = \{11,100,001,0000\}$
  - $L^3 = \{111,1100,1001,10000,000000,00001,00100,0011\}$
  - $\mathbf{L^*} = L^0 \cup L^1 \cup L^2 \cup \ldots$

Context-Free Grammar CFG

- The structure of the rules known as *Context-Free Grammar* CFG.
- The process of generating of final string of leaves starting from the beginning of a sequence of rules is known as *derivation*.
- The language which is generated by CFG is called *Context-free Language*.

Context-Free Grammar CFG

- Context-Free Grammar is a collection of three items:
  - An alphabet of letters called *terminals* and r.
  - A group of symbols and known as non-*terminals* one of them act as a start symbol.
  - A sequence of *productions* of the form of:

Non-terminal → string of terminals and/or non-terminals

Context-Free Grammar CFG

- A ***Context Free Grammar*** is a “machine” that creates a language.
- A language created by a CF grammar is called ***A Context Free Language***.

.

Context-Free Grammar CFG

Consider grammar G_1 : $I \rightarrow aIb$

$I \rightarrow \varepsilon$

A CFL consists of substitution rules called ***Productions***.

The capital letters are the ***Variables***.

The other symbols are the ***Terminals***

Context-Free Grammar CFG

The grammar G_1 ***generates*** the language

$B = \{a^n b^n \mid n \geq 0\}$ called ***the language of*** G_1

denoted by $L(G_1)$

This is a ***Derivation*** of the word $aaabbb$ by G_1

$$I \Rightarrow aIb \Rightarrow aaIbb \Rightarrow aaaIbbb \Rightarrow aaabbb$$

Chomsky Normal Form (CNF)

Let G be a CFG for some L-{ε}

<u>Definition:</u>

G is said to be in ***Chomsky Normal Form*** *if all its productions are in one of the following two forms:*

i. ***A → BC*** *where A,B,C are variables, or*

ii. ***A → a*** *where a is a terminal*

– *G has no useless symbols*

– *G has no unit productions*

– *G has no ε-productions*

Chomsky Normal Form (CNF)

- Example

G:

1. E → E+T | T*F | (E) | Ia | Ib | I0 | I1
2. T → T*F | (E) | Ia | Ib | I0 | I1
3. F → (E) | Ia | Ib | I0 | I1
4. I → a | b | Ia | Ib | I0 | I1

CFGs & ambiguity

- Ambiguity of CFGs
  - To show that a CFG is ambiguous, given one input string in the language which has more than one parse tree
    - (or equivalenty, >1 leftmost/rightmost derivation)
  - Finding one example is sufficient

- A CFL is *inherently ambiguous* if all grammars for that language are going to be ambiguous

- Converting ambiguous CFGs to non-ambiguous CFGs
  - Not possible for inherently ambiguous CFLs
  - For unambiguous CFLs, use ambiguity resolving techniques (e.g., precedence)

Finite Automata

- Deterministic Finite Automata (DFA)
  - The machine can exist in only one state at any given time
- Non-deterministic Finite Automata (NFA)
  - The machine can exist in multiple states at the same time
- ε-NFA is an NFA that allows ε-transitions
- What are their differences?

Deterministic Finite Automata

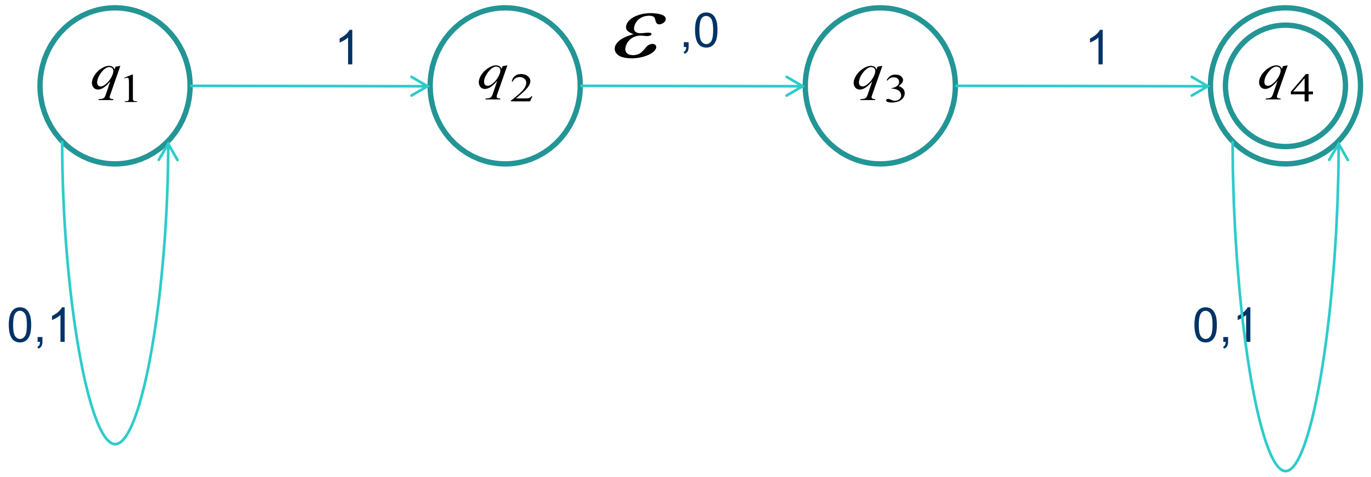
- A DFA is defined by the 5-tuple:
  - $\{Q, \Sigma, q_0, F, \delta\}$
- Two ways to define:
  - State-diagram (preferred)
  - State-transition table
- DFA construction checklist:
  - Associate states with their meanings
  - Capture all possible combinations/input scenarios
    - break into cases & sub cases wherever possible
  - Are outgoing transitions defined for every symbol from every state?
  - Are final/accepting states marked?
  - Possibly, dead/error-states will have to be included depending on the design.

Non Deterministic Finite Automata

- A NFA is defined by the 5-tuple:
  - $\{Q, \Sigma, q_0, F, \delta\}$
- Two ways to represent:
  - State-diagram (preferred)
  - State-transition table
- NFA construction checklist:
  - Has *at least* one nondeterministic transition
  - Capture only valid input transitions
    - Can ignore invalid input symbol transitions (paths will die implicitly)
  - Outgoing transitions defined only for valid symbols from every state
  - Are final/accepting states marked?

Non Deterministic Finite Automata

- NFA – Nondeterministic Finite Automaton



$$\begin{tikzcd}
q_1 \arrow[r, "1"] \arrow[loop below, "{0,1}"] & q_2 \arrow[r, "{\varepsilon,0}"] & q_3 \arrow[r, "1"] & q_4 \arrow[loop below, "{0,1}"]
\end{tikzcd}$$

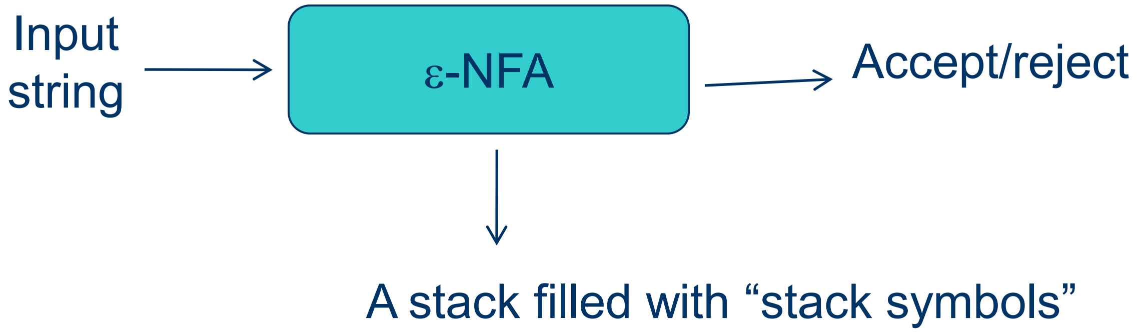
1. A state nay have 0 or more transitions labeled with the same symbol.
2. ε transitions are possible.

Regular Expressions

- DFA to Regular expression
  - Enumerate all paths from start to every final state
  - Generate regular expression for each segment, and concatenate
  - Combine the reg. exp. for all each path using the + operator
- Reg. Expression to ε -NFA conversion
  - Inside-to-outside construction
  - Start making states for every atomic unit of RE
  - Combine using: concatenation, + and * operators as appropriate
  - For connecting adjacent parts, use ε -transitions
  - Remember to note down final states

Pushdown Automata (PDA)

- What is?
  - FA to Reg Lang,   PDA is to CFL
- PDA == [ε -NFA + “a stack”]
- Why a stack?



Pushdown Automata - Definition

- A PDA P := (Q,∑,Γ, δ,q_0,Z_0,F):

– Q:	states of the ε-NFA
– ∑:	input alphabet
– Γ :	stack symbols
– δ:	transition function
– q_0:	start state
– Z_0:	Initial stack top symbol
– F:	Final/accepting states

Example

Let $L_{wwr} = \{ww^R \mid w \text{ is in } (0+1)^*\}$

- CFG for L_{wwr} : $S \Longrightarrow 0S0 \mid 1S1 \mid \varepsilon$
- PDA for L_{wwr} :
- $P := (Q, \sum, \Gamma, \delta, q_0, Z_0, F)$

$= (\{q_0, q_1, q_2\}, \{0,1\}, \{0,1,Z_0\}, \delta, q_0, Z_0, \{q_2\})$

Turing Machines

- Very powerful (abstract) machines that could simulate any modern day computer (although very, very slowly!)
- Why design such a machine?
  - If a problem cannot be "solved" even using a TM, then it implies that the problem is ***undecidable***
- Computability vs. Decidability