Theory of computation is the theoretical study of capabilities and limitations of Computers (Theoretical models of computation).

Objectives:

Providing students with:

- an understanding of basic concepts in the theory of computation through simple models of computational devices.
- apply models in practice to solving problems in diverse areas such as string searching, pattern matching, cryptography, and language design;
- understand the limitations of computing, the relative power of formal languages and the inherent complexity of many computational problems.
- be familiar with standard tools and notation for formal reasoning about machines and programs.

REFERENCES:

1. Introduction to Computer Theory 2nd Edition, Daniel I. A. Cohen John Wiley & Sons, Inc 1997. ISBN 0-471-13772-3

2. Introduction to Automata Theory, Languages, and Computation, 2/E, John E. Hopcroft, Rajeev Motwani, Jeffrey D.Ullman, Addison-Wesley 2001. ISBN 0-201-44124-1.

As a computer student, you must study the following:

1- Automata and formal language.

Which answers - What are computers (Or what are models of computers)

2- Compatibility.

Which answers - What can be computed by computers?

3- Complexity.

Which answers - What can be efficiently computed?study of capabilities and limitations of Computers

In automata we will simulate parts of computers. Or we will make mathematical models of computers. Automata are more powerful than any real computer because we can design any machine on papers that can do everything we want.

Theory of computation is the theoretical study of capabilities and limitations of Computers

(Theoretical models of computation).

Sets

Let A, B, and C be subsets of the universal set U

Distributive properties

$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

Theory of computation is the theoretical $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

Idempotent properties

$A \cap A = A$

$A \cup A = A$

Double Complement property

$(A^{-})^{-} = A$

De Morgan’s laws

$(A \cup B)^{-} = A^{-} \cap B^{-}$

$(A \cap B)^{-} = A^{-} \cup B^{-}$

Commutative properties

$A \cap B = B \cap A$

$A \cup B = B \cup A$

Associative laws

$A \cap (B \cap C) = (A \cap B) \cap C$

$A \cup (B \cup C) = (A \cup B) \cup C$

Identity properties

$A \cup \varnothing = A$

$A \cap u = A$

Complement properties

$A \cup \overline{A} = u$

$A \cap \overline{A} = \varnothing$

Language

language is the set of all strings of terminal symbols derivable from the alphabet.

The alphabet is a finite set of symbols. For example {0, 1} is an alphabet with two symbols, {a, b} is another alphabet with two symbols and the English alphabet is also an alphabet.

A **string** (also called a word) is a finite sequence of symbols of an alphabet. b, a and aabab are examples of string over alphabet {a, b} and 0, 10 and 001 are examples of string over alphabet {0, 1}, A null string is a string with no symbols, usually denoted by epsilon or lambda (λ).

A **language** is a set of strings over an alphabet. Thus {a, ab, baa} is a language (over alphabert {a,b}) and {0, 111} is a language (over alphabet {0,1}). The number of symbols in a string is called the **length of the string**. For a string w its length is represented by |w| . It can be **The empty string** (also called null string) it has no symbols. The empty string is denoted by λ Thus $|\lambda| = 0$.

For example |00100| = 5, |aab| = 3, $|\lambda| = 0$

Language = alphabet + string (word) + grammar (rules, syntax) + operations on languages (concatenation, union, intersection, Kleene star)

Kinds of languages:

1- Talking language: (e.g.: English, Arabic): It has alphabet: ∑= {a,b,c,….z} From these alphabetic we make sentences that belong to the language. Now we want to know if this sentence is true or false so we need grammar. Ali is a clever student. (It is a sentence @ English language.)

2- Programming language: (e.g.: c++, Pascal):It has alphabetic:

∑= {a,b,c,.z , A,B,C,..Z , ?, /, - ,\.}

From these letters we make sentences that belong to the programming language. Now we want to know if this sentence is true or false so we need a compiler to make sure that syntax is true.

3- Formal language: (any language we want.) It has strings from these strings we make sentences that belong to this formal language.

Now we want to know if this sentence is true or false so we need rules.

Example:

Alphabetic: ∑= {0, 1}.

Sentences: 0000001, 1010101.

Rules: Accept any sentence starting with zero and refuse sentences that start with one.

So we **accept**: 0000001 as a sentence that satisfies the rules.

And **refuse**: 1010101 as a sentence doesn't satisfy the rules.

Example:

Alphabetic: ∑= {a, b}.

Sentences: ababaabb, bababbabb

Rules: Accept any sentence starting with a and refuse sentences that start with b.

So we **accept**: aaaaabba as a sentence satisfies the rules.

And **refuse**: baabbaab as a sentence doesn't satisfy the rules.

Regular Expression

Regular Expression is a set of symbols, Thus if alphabet= {a, b}, then aab, a, baba, bbbbb, and baaaaa would all be strings of symbols of the alphabet.

In addition we include an empty string denoted by which has no symbols in it.

Examples of Kleene star:

1* is the set of strings { λ , 1, 11, 111, 1111, 11111, etc. }

(1100)* is the set of strings { λ , 1100, 11001100, 110011001100, etc. }

(00+11)* is the set of strings {epsilon, 00, 11, 0000, 0011, 1100, 1111, 000000, 000011, 001100, etc. }

(0+1)* is all possible strings of zeros and ones, often written as sigma * where sigma = {0, 1}

(0+1)* (00+11) is all strings of zeros and ones that end with either 00 or 11.

(w)+ is a shorthand for (w)(w)* w is any string or expression and the superscript plus, +

1- Concatenation:

Notation to the concatenation: . (The dot.):

if L1 = {x, xxx} and L2 = {xx} So (L1.L2) means L1 concatenated L2 and it is equal = {xxx, xxxxx}

Examples on concatenations:

Ex1:

L1 = {a, b}.

L2 = {c, d}.

L1.L2 = {ac, ad, bC, bd}

Note: ab differ from ba.

Ex2:

∑ = {x}.

L1 = {set of all odd words over with odd length}.

L1 = {set of all even words over with odd length}.

L1= {x, xxx, xxxxx, xxxxxxx……}.

L2= { , xx, xxxx, xxxxxx…}.

L1.L2 = {x, xxx, xxxxx, xxxxxxx…}.

Note:

التكرار غير مسموح داخل المجموعة

Ex3:

L1 = {x, xxx}.

L2 = {xx}.

L1.L2 = {xxx, xxxxx}.

Some rules on concatenation:

λ.x = x

L1.L2 = {set of elements}

Definition of a Regular Expression

- A regular expression may be the null string.
  r = λ
- A regular expression may be an element of the input alphabet,
  r = a
- A regular expression may be the union of two regular expressions,
  r = r1+r2
- A regular expression may be the concatenation of two regular expressions. r = r1r2
- A regular expression may be the Kleene closure (star) of a regular expression. r = r1*
- A regular expression may be a regular expression in parenthesis
  r = (r1)

Epsilon is the zero length string

0, 1, a, b, c, are symbols in sigma

x is a variable or regular expression

(...)(...) is concatenation

(...) + (...) is union

(...)* is the Kleene Closure = **Kleene Star**

(λ)(x) = (x)(λ) = λ

(λ)(x) = (x)(λ) = x

(λ) + (x) = (x) + (λ) = x

x + x = x

(λ)* = (λ)(λ) = λ

(x)* + (λ) = (x)* = x*

(x + λ)* = x*

x* (a+b) + (a+b) = x* (a+b)

$x^* y + y = x^* y$

$(x + \lambda) x^* = x^* (x + \lambda) = x^*$

$(x+ \lambda)(x+ \lambda)^* (x+ \lambda) = x^*$

λ is the null string (there are no symbols in this string)

* is the set of all strings of length greater than or equal to 0

Example:

A = {a,b} // the alphabet is composed of a and b

$A^* = \{\lambda, a,b,aa,ab,ba,bb,aaa,aab,\ldots\}$

The symbol * is called the Kleene star.

$\varnothing$ (empty set)

λ (empty string)

( ) delimiter ,

U + union (selection) concatenation

Example 1

Let A={0,1}, W1 = 00110011, W2 = 00000

W1W2 = 0011001100000

W2W1 = 0000000110011

$W1\ \lambda = W1 = 00110011$

$\lambda\ W2 = W2 = 00000$

Note:

$(a + b)^* = (a^*b^*)^*$

Regular Expansion	output (sets of strings)
λ	{λ}
λ^*	{λ}
a	{ a }
aa	{ aa }
a*	{ λ, a, aa, aaa, … }
aa*	{ a, aa, aaa, … }
(aa)*	{ aa, aaaa, aaaaaa, … }
a^+	{ a, aa, aaa, … }
ba^+	{ ba, baa, baaa, … }
$(ba)^+$	{ ba, baba, bababa, … }
(a | b)	{ a, b }
a|b*	{ a, λ, b, bb, bbb, … }
(a | b)*	{ λ, a, b, aa, ab, ba, bb, … }
aa(ba)*bb	{ aabb, aababb, aababababb, … }
(a + a)	{ a }
(a + b)	{ a, b}
(a + b)(a + b)	{aa, ab, ba, bb}
(a + b + c)	{a, b, c}
(a + b)*	{ λ, a, b, aa, bb, ab, ba, aaa, bbb, aab, bba, … }
(abc)	{abc}
(λ + a) bc	{bc, abc}
ab*	{a. ab, abb, abbb, …}
(ab)*	{ λ, ab, abab, ababab, … }

a + b*	{ a, λ, b, bb, bbb, … }
a (a + b)*	{ a,aa, ab, aaa, abb, aba, abaa, … }
(a + b)* a (a + b)*	{a, aaa, aab, baa, bab, … }
(a + λ)*	(a)* = { λ, a, aa, aaa, … }
x* (a + b) + (a + b)	x* (a + b)
x* (y + y)	x*y
(x + λ) x*	= x* = {λ, x, xx, xxx, …}
(x + λ)(x + λ)* (x + λ)	x*
start with a	a (a+b)*
end with b	(a + b)* b
start with a and end with b	a (a + b)* b
start with a or b	(a + b)(a + b)*
not start with b	a (a + b)*
contains exactly 2 a's	(b)* a (b)* a (b)*
contains at least 2 a's	(a + b)* a (a + b)* a (a+ b)*
contains exactly 2 a's or 2 b's	[(b)* a (b)* a (b)*] + [(a)* b (a)* b (a)*]
contains even no of a	[(b)* a (b)* a (b)*]*
not start with a and not contain b	{λ}
with even length of a	$(aa)^+$
strings containing 101	(1 + 0)* 101 (1 + 0)*
even number of 0's and contains 11	(00)* 11 (00)*
even number of 0's or contains 11	(00)* + (0* 11 0*)
every 1 has at least two 0	0* $(100)^+$ 0*

that follow it	
second symbol not a one	$(1 + 0)\ 0\ (1 + 0)^*$
end with 00 or 01	(1 + 0) (00 + 01)

Exercise

Ex.1: Find a regular expression over the alphabet { a, b } that contains exactly three a's.

Ex.2: Find a regular expression over the alphabet { a, b } that ends with ab.

Ex.3: Find a regular expression over the alphabet { a, b } that has length of 3.

Ex. 4: Find a regular expression over the alphabet { a, b } that contains exactly two successive a's.

Ex. 5: Find the output (words) for the following regular expr

$(\lambda)^*$	
$(x)^* + (\lambda)$	
aa^*b	
bba^*a	
$(a + b)^*\ ba$	
$(0 + 1)^*\ 00\ (0 + 1)^*$	
$(11 + 0)^*\ (0 + 11)^*$	
$01^* + (00 + 101)^*$	
$(a + b)^*\ abb^+$	
$(\ (\ (01 + 10)^*\ 11)^*\ 00)^*$	