Chapter 1: Introduction

Introduction

Purpose of the Theory of Computation: Develop formal mathematical models of computation that reflect real-world computers. Nowadays, the Theory of Computation can be divided into the following three areas:

- Automata Theory
- Computability Theory
- Complexity Theory,

Automata theory

Automata Theory deals with definitions and properties of different types of "computation models". Examples of such models are:

- Finite Automata. These are used in text processing, compilers, and hardware design.
- Context-Free Grammars. These are used to define programming languages and in Artificial Intelligence.
- Turing Machines. These form a simple abstract model of a "real" computer, such as your PC at home.

Central Question in Automata Theory: Do these models have the same power, or can one model solve more problems than the other?

Computability theory

In the 1930's, G¨odel, Turing, and Church discovered that some of the fundamental mathematical problems cannot be solved by a "computer". (This may sound strange, because computers were invented only in the 1940's).

An example of such a problem is "Is an arbitrary mathematical statement true or false?" To attack such a problem, we need formal definitions of the notions of

- computer,
- algorithm, and
- computation.

The theoretical models that were proposed in order to understand solvable and unsolvable problems led to the development of real computers.

Central Question in Computability Theory: Classify problems as being solvable or unsolvable.

Complexity Theory

The main question asked in this area is "What makes some problems computationally hard and other problems easy?"

Informally, a problem is called "easy", if it is efficiently solvable. Examples of "easy" problems are (i) sorting a sequence of, say, 1,000,000 numbers, (ii) searching for a name in a telephone directory, and (iii) computing the fastest way to drive from Ottawa to Miami. On the other hand, a problem is called “hard”, if it cannot be solved efficiently, or if we don’t know whether it can be solved efficiently. Examples of “hard” problems are (i) time table scheduling for all courses at Carleton, (ii) factoring a 300-digit integer into its prime factors, and (iii) computing a layout for chips in VLSI.

***Central Question in Complexity Theory**: Classify problems according to their degree of “difficulty”. Give a rigorous proof that problems that seem to be “hard” are really “hard”.*

This course

- This course is about the fundamental capabilities and limitations of computers. These topics form the core of computer science.
- It is about mathematical properties of computer hardware and software.
- 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., etc.
- This course helps you to learn problem solving skills. Theory teaches you how to think, prove, argue, solve problems, express, and abstract.
- This theory simplifies the complex computers to an abstract and simple mathematical model, and helps you to understand them better.
- This course is about rigorously analyzing capabilities and limitation of systems.

Turing machine is equivalent in computing power to the digital computer as we know it today and also to all the most general mathematical notions of computation

1.1 Introduction to Set Theory

Set

A set is a collection of objects. For example, the collection of the four letters a, b, c, and d is a set, which we may name L; we write L = {a, b, c, d}. The objects comprising a set are called its elements or members. For example, b is an element of the set L; in symbols, b ϵ L. Sometimes we simply say that b is in L, or that L contains b. On the other hand, z is not an element of L, and we write z ϵ L.

Singleton set

A set may have only one element; it is then called a singleton. For example, {1} is the set with 1 as its only element; thus {1} and 1 are quite different.

Empty set

There is also a set with no element at all. Naturally, there can be only one such set: it is called the empty set, and is denoted by Ø. Any set other than the empty set is said to be nonempty.

<u>1.2 Set Operations</u>

Union

That is, the union of sets *A* and *B*, written $A \cup B$, is a set that contains everything in *A*, or in *B*, or in both.

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

Intersection

The "intersection" of sets *A* and *B*, written $A \cap B$, is a set that contains exactly those elements that are in both *A* and *B*.

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

Set Difference

The "set difference" of set *A* and set *B*, written as *A*–*B*, is the set that contains everything that is in *A* but not in *B*.

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

Complement

The "complement" of set A, written as A is the set containing everything that is not in A.

Properties of set operations

Idempotency:	$A\cup A= A$
	$A \cap A= A$
Commutativity :	$A\cup B =B\cup A$
	$A\cap B = B\cap A$
Associativity :	$(A\cup B) \cup C= A\cup (B \cup C)$
	$(A\cap B) \cap C = A \cap (B \cap C)$
Distributivity :	$(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$
	$(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$
Absorption :	$(A \cup B) \cap A = A$
	$(A \cap B) \cup A = A$
DeMorgan's Laws :	$A - (B \cup C) = (A - B) \cap (A - C)$
	$A - (B \cap C) = (A - B) \cup (A - C)$

Show that $A - (B \cup C) = (A - B) \cap (A - C)$.

$$\begin{aligned} x \in A-(B\cup C) &\Rightarrow x\in A \text{ and } x\notin B\cup C \\ &\Rightarrow x\in A \text{ and } x\notin B \text{ and } x\notin C \\ &\Rightarrow (x\in A \text{ and } x\notin B) \text{ and } (x\in A \text{ and } x\notin C) \\ &\Rightarrow x\in A-B \text{ and } x\in A-C \\ &\Rightarrow x\in (A-B)\cap(A-C) \end{aligned}$$

Therefore $A-(B\cup C)\subseteq(A-B)\cap(A-C)$ (1)

Conversely,

$$\begin{aligned} x \in (A-B)\cap(A-C) &\Rightarrow x\in A-B \text{ and } x\in A-C \\ &\Rightarrow (x\in A \text{ and } x\notin B) \text{ and } (x\in A \text{ and } x\notin C) \\ &\Rightarrow x\in A \text{ and } (x\notin B \text{ and } x\notin C) \\ &\Rightarrow x\in A \text{ and } x\notin B\cup C \\ &\Rightarrow x\in A-(B\cup C) \end{aligned}$$

Therefore, $(A-B)\cap(A-C)\subseteq A-(B\cup C)$.

Hence $A-(B\cup C)=(A-B)\cap(A-C)$.

Additional Terminology

(a) Disjoint Sets. If *A* and *B* have no common element, that is, $A \cap B = ø$, then the sets *A* and *B* are said to be disjoint.

(b) Cardinality. The "Cardinality" of a set *A*, written |*A*|, is the number of elements in set *A*.

(c) Powerset. The "powerset" of a set *A*, written 2*A*, is the set of all subsets of *A*; i.e., a set containing '*n*' elements has a powerset containing 2*n* elements.

(d) Cartesian Product. Let *A* and *B* be two sets. Then the set of all ordered pairs (*x*, *y*) where *x* ϵ *A* and *y* ϵ *B* is called the "Cartesian Product" of the sets *A* and *B* and is denoted by *A* x *B*, i.e.
A x *B* = { (*x*, *y*) : *x* ϵ *A* and *y* ϵ *B* }

1.3 Relations and Functions

Definition of Relation: A relation on sets S and T is a set of ordered pairs (s, t), where
(a) s ϵ S (s is a member of S)
(b) t ϵ T
(c) S and T need not be different
(d) The set of all first elements in the "domain" of the relation, and
(e) The set of all second elements is the "range" of the relation.

Example:

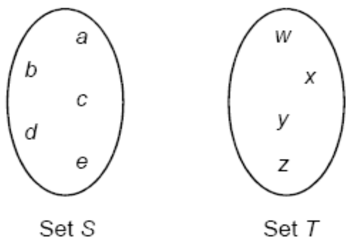


Fig. 1 Sets S and T are disjoint

Suppose S is the set $\{a, b, c, d, e\}$ and set T is $\{w, x, y, z\}$.

Then a relation on S and T is

$$R = \{(a, y), (c, w), (c, z), (d, y)\}$$

The four ordered pairs in the relation is represented as shown in Fig. 2.

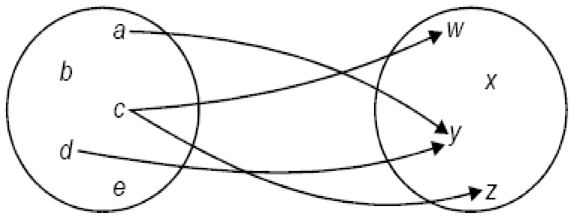


Fig. 2 Relation $R = \{(a, y), (c, w), (c, w), (c, z), (d, y)\}$

Types of Relations

1. Reflexive Relation
2. Symmetric/Anti- symmetric relation
3. Transitive relation
4. Equivalence Relation
5. Partial order/Total Order Relation

Reflexive

A relation $R \subseteq A \times A$ is reflexive if $(a, a) \in R$ for each $a \in A$. The directed graph representing a reflexive relation has a loop from each node to itself.

Let A= { 1, 2, 3}

then R= { (1, 1), (2, 2) , (3, 3) } is a reflexive relation defined on set A

Symmetric

A relation $R \subseteq A \times A$ is symmetric if $(b, a) \in R$ whenever $(a, b) \in R$.

Let A= { 1, 2, 3}

then R= { (1, 2), (2, 1) , (2, 3) , (3, 2) } is a Symmetric relation defined on set A

Note: if the relation is not symmetric then it is anti -symmetric

Transitive

A binary relation R is transitive if whenever (a, b) ∈ R and (b, c) ∈ R, then (a, c) ∈ R. The relation {(a, b) : a, b ∈ P and a is an ancestor of b} is transitive, since if a is an ancestor of band b is an ancestor of c, then a is an ancestor of c. So is the less-than-or-equal relation
Let A= { 1, 2, 3}
then R= { (1, 2), (2, 3) , (1, 3) } is a transitive relation defined on set A

Equivalence Relation

A subset *R* of *A* x *A* is called an equivalence relation on *A* if *R* satisfies the following conditions:
(i) (*a*, *a*) ϵ *R* for all *a* ϵ *A* (*R* is reflexive)
(ii) If (*a*, *b*) ϵ *R*, then (*b*, *a*) ϵ *R*, then (*a*, *b*) ϵ *R* (*R* is symmetric)
(iii) If (*a*, *b*) ϵ *R* and (*b*, *c*) ϵ *R*, then (*a*, *c*) ϵ *R* (*R* is transitive)

Let A= { 1, 2, 3}
then R= { (1, 1), (2, 2) , (3, 3) , (1, 2) , (1, 3) , (2, 3) , (2, 1) , (3, 1) , (3, 2) }

Partial Ordering Relations

A relation *R* on a set *S* is called a "Partial ordering" or a "Partial order", if *R* is reflexive, anti-symmetric and transitive.
Let A= { 1, 2, 3}
then R= { (1, 1), (2, 2) , (3, 3) , (1, 2), (2, 3) }

A set *S* together with a partial ordering *R* is called a "Partially ordered set" or "Poset".

Partition

A Partition *P* of *S* is a collection {*Ai*} of nonempty subsets of *S* with the properties:
(i) Each *a* ϵ *S* belongs to some *Ai*,
(ii) If *Ai* ≠ *Aj*, then *Ai* ∩ *Aj* = ø.

Functions

Suppose every element of S occurs exactly once as the first element of an ordered pair. In Fig shown, every element of S has exactly one arrow arising from it. This kind of relation is called a "function".

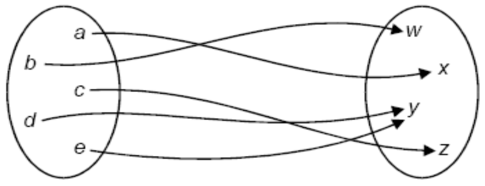


Fig. A Function

A function is otherwise known as "Mapping". A function is said to map an element in its domain to an element in its range. Every element in S in the domain, i.e., every element of S is mapped to some element in the range. No element in the domain maps to more than one element in the range.

Functions as relations

A function $f : A \rightarrow B$ is a relation from A to B i.e., a subset of A x B, such that each $a \in A$ belongs to a unique ordered pair (a, b) in f.

Kinds of Functions

(a) *One-to-One Function* (*Injection*): A function $f : A \rightarrow B$ is said to be one-to-one if different elements in the domain A have distinct images in the range.

A function f is one-to-one if $f(a) = f(a')$ implies $a = a'$.

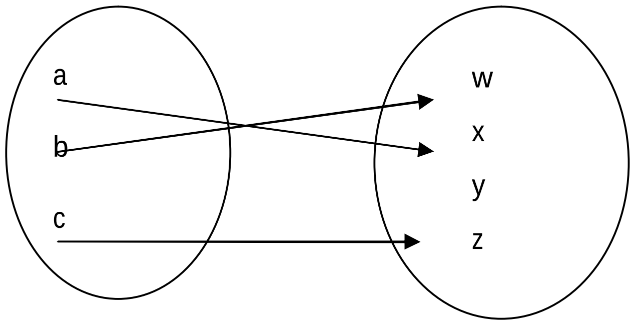


Fig: One to one function (Injection)

(b) *Onto function (Surjection): A function $f : A \rightarrow B$ is said to be an onto function if each element of B* is the image of some element of *A*. i.e., $f : A \rightarrow B$ is onto if the image of f is the entire codomain, i.e. if $f(A) = B$. i.e., f maps A onto B.

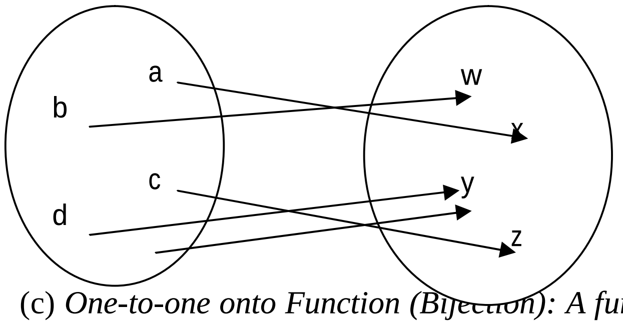


Fig: Surjection

(c) *One-to-one onto Function (Bijection): A function that is both one-to-one and onto is called a "Bijection". Such a function maps each and every element of A* to exactly one element of *B*, with no elements left over. Fig. below shows bijection.

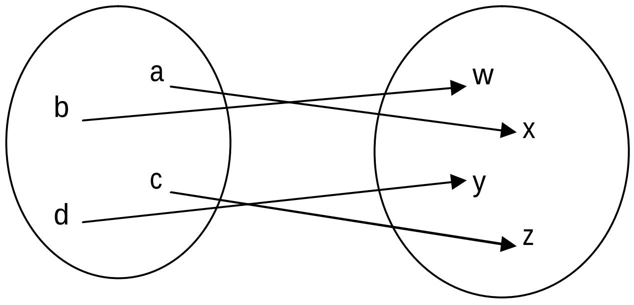


Fig: *Bijection*

1.3 Fundamental Proof Techniques

1. **Induction Principle**
2. **Diagonalization Principle**
3. **Pigeonhole Principle**

1. Induction Principle

The principle of mathematical induction states that any set of natural numbers containing zero, and with the property that it contains $n + 1$ whenever it contains all the numbers up to and including n, must in fact be the set of all natural numbers.

Let we want to show that property P holds for all natural numbers. To prove this property, P using mathematical induction following are the steps:

Basic Step:

First show that property P is true for 0 or 1

Induction Hypothesis:

Assume that property P holds for n

Induction Step:

Using induction hypothesis, show that P is true for n+1

Then by the principle of mathematical induction, P is true for all natural numbers

Example

Let us show that for any $n \geq 0$, 1+2+3+…………………+ n $= \frac{m2+1}{2}$

Basic Step. Let n=0. Then the sum on the left is zero, since there is nothing to add. The expression on the right is also zero.

Induction hypothesis:

Assume that ,for some $m \geq 0$,1+2+3+…………………+ m $= \frac{m2+1}{2}$

Induction Step.

$$\begin{aligned} 1 + 2 + \cdots + n + (n+1) &= (1 + 2 + \cdots + n) + (n+1) \\ &= \frac{n^2+n}{2} + (n+1) \quad \text{(by the induction hypothesis)} \\ &= \frac{n^2+n+2n+2}{2} \\ &= \frac{(n+1)^2 + (n+1)}{2} \end{aligned}$$

Examples

Example 1:

Prove using mathematical induction, $n^4 - 4n^2$ is divisible by 3 for n>=0.

Basic step:

For n=0,

$n^4 - 4n^2$ =0, which is divisible by 3.

Induction hypothesis:

Let $n^4 - 4n^2$ is divisible by 3.

Induction step:

$(n+1)^4 - 4(n+1)^2$

$= [(n+1)^2]^2 - 4(n+1)^2$

$= (n^2+2n+1)^2 - (2n+2)^2$

$= (n^2+2n+1+2n+2)(n^2+2n+1-2n-2)$

$= (n^2+4n+3)(n^2-1)$

$= n^4+4n^3+3n^2-3-4n-n^2$

$= n^4+4n^3+2n^2-4n-3$

$= n^4+4n^3-4n^2+6n^2-4n-3$

$= n - 4n^2 + 6n^2 - 3 + 4n^3 - 4n$

$= (n^2 - 4n^2) + (6n^2) - (3) + 4(n^3 - n)$

$(n^2 - 4n^2)$ is divisible by 3 from our hypothesis.

$6n^2, 3$ are divisible by 3.

We need to prove that $4(n^3 - n)$ is divisible by 3.

Again use mathematical induction.

Basic step:

For n = 0,

4(0-0) = 0 is divisible by 3.

Induction hypothesis:

Let $4(n^3 - n)$ is divisible by 3.

Induction step:

$4[(n+1)^3 - (n+1)]$

$= 4[(n^3 + 3n^2 + 3n + 1) - (n+1)]$

$= 4[n^3 + 3n^2 + 3n + 1 - n - 1]$

$= 4[n^3 + 3n^2 + 2n]$

$= 4[n^3 - n + 3n^2 + 3n]$

$= 4(n^3 - n) + 4.3n^2 + 4.3n$

$4(n^3 - n)$ is divisible by 3 from our hypothesis.

$4.3n^2$ is divisible by 3.

$4.3n$ is divisible by 3.

Thus we can say that

$= (n^2 - 4n^2) + (6n^2) - (3) + 4(n^3 - n)$ is divisible by 3.

That is,

$n^4 - 4n^2$ is divisible by 3.

Example 2:

Prove using mathematical induction:

$1+2+3+.......+n=\frac{n(n+1)}{2}$

Let $P(n)=1+2+3+.......+n=\frac{n(n+1)}{2}$

Basic Step:

For n=1,

LHS = 1

RHS = 1(1+1)/2=1

Induction hypothesis;

Assume that P(n) is true for n=k,

Then,

$1+2+3+.......+k=\frac{k(k+1)}{2}$

Induction step:

Show that P(n) is true for n=k+1.

$1+2+3+.......+k+(k+1)$

$=\frac{k(k+1)}{2}+k+1$

$=(k+1)[\frac{k}{2}+1]$

$=(k+1)(\frac{k+2}{2})$

$=\frac{(k+1)(k+2)}{2}$

That is, P(n) is true for n=k+1.

Thus by using the principle of mathematical induction, we proved

$1+2+3+.......+n=\frac{n(n+1)}{2}$

Example 3:

Prove using the principle of mathematical induction,

$\sum_{i=0}^{n} n^2 = \frac{n(n+1)(2n+1)}{6}$

Let $P(n) = \sum_{i=0}^{n} n^2 = \frac{n(n+1)(2n+1)}{6}$

Basic step:

For n=1,

LHS = $1^2 = 1$

RHS = $\frac{n(n+1)(2n+1)}{6} = \frac{1.2.3}{6} = 1$

P(n) is true for n=1.

Induction hypothesis:

Assume that result is true for n=k.

That is,

$1^2 + 2^2 + ... + k^2 = \frac{k(k+1)(2k+1)}{6}$

Induction step:

Prove that result is true for n=k+1.

$1^2 + 2^2 + ... + k^2 + (k+1)^2 = \frac{k(k+1)(2k+1)}{6} + (k+1)^2$

$= (k+1)[\frac{k(2k+1)}{6} + (k+1)]$

$= (k+1)(\frac{2k^2+k+6k+6}{6})$

$= (k+1)(\frac{2k^2+7k+6}{6})$

$= (k+1)(\frac{2k^2+4k+3k+6}{6})$

$= (k+1)(\frac{2k(k+2)+3(k+2)}{6})$

$= \frac{(k+1)(k+2)(2k+3)}{6}$

Thus it is proved.

Diagonalization Principle

Let R be a binary relation on a set A, and let D, the diagonal set for R, be {a : a ϵ A and (a, a) ∉ R}. For each a ϵ A, let Ra = {b : b ϵ A and (a, b) ϵ R}. Then D is distinct from each Ra.

Let s be a non empty set and R any relation on S.

Let

$D = \{a \epsilon A | (a, a) \notin r\}$

For each $a \notin A$, let $R_a = \{b | (a, b) \in R\}$

Then diagonalization principle states that D is different from each R_a.

OR

Diagonalization principle states that the complement of the diagonal is different from ea

For example,

Let S = {a, b, c, d}

R = { (a,a), (b,c), (b,d), (c,a), (c,c), (c,d), (d,a), (d,b) }

The above relation R is shown in matrix from as follows:

$$\begin{array}{c|c|c|c|c|}
 & a & b & c & d \\
\hline
a & \boxed{X} & & & \\
\hline
b & & \boxed{} & X & X \\
\hline
c & X & & \boxed{X} & X \\
\hline
d & X & X & & \boxed{} \\
\hline
\end{array}$$

Diagonal elements are marked.

From the figure, $R_a = \{a\}$

$R_b = \{c, d\}$

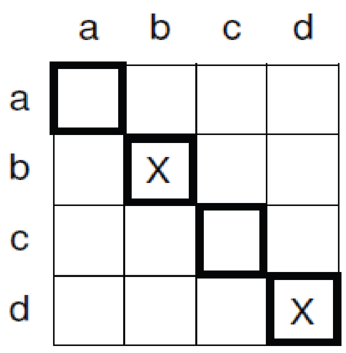
$R_c = \{a, c, d\}$

$R_d = \{a, b\}$

Complement of the diagonal is,

D = {b, d}

That is,



If we compare each of the above R_a, R_b, R_c, R_d with D, we can see that D is different from each R_a. Thus complement of the diagonal is distinct from each row.

Pigeonhole Principle

If A and B are finite sets and |A| > |B|, then there is no one-to-one function from A to B. i.e., If an attempt is made to pair off the elements of A (the “pigeons”) with elements of B (the “pigeonholes”), sooner or later we will have to put more than one pigeon in a pigeonhole.
The pigeonhole principle states that if n pigeons are put into m pigeonholes with n>m, then at least one pigeonhole must contain more than one pigeon.
Thus if S1 and S2 are two non empty finite sets and |S1| > |S2|, then there is no one-to-one function from S1 to S2. Pigeonhole principle can be used to show that certain languages are not regular. We will use pigeonhole principle in the topic pumping lemma later.

1.4 Some Terminologies

Alphabets :
An alphabet is a finite, nonempty set of symbols. The alphabet of a language is normally denoted by Σ .
Example :

$\Sigma = \{0, 1\}$

$\Sigma = \{a, b, c\}$

$\Sigma = \{a, b, c, \&, z\}$

$\Sigma = \{\#, \nabla, \spadesuit, \beta\}$

Strings or Words over Alphabet :
A string or word over an alphabet Σ is a finite sequence of concatenated symbols of Σ. Denoted by Σ*
Example : 0110, 11, 001 are three strings over the binary alphabet { 0, 1 } .
aab, abcb, b, cc are four strings over the alphabet { a, b, c }.

It is not the case that a string over some alphabet should contain all the symbols from the alphabet. For example, the string cc over the alphabet { a, b, c } does not contain the symbols a and b. Hence, it is true that a string over an alphabet is also a string over any superset of that alphabet.

Length of a string :
The number of symbols in a string w is called its length, denoted by |w|.

Example : | 011 | = 4, |11| = 2, | b | = 1

It is convenient to introduce a notation e for the empty string, which contains no symbols at all. The length of the empty string e is zero, i.e., | e | = 0.

Empty string:
The string of zero length is empty string. It is denoted by e or λ.

Reverse string:
IF w= $w_1w_2 \ldots w_n$ where each $w_i \in \Sigma$, the reverse of w is $w_n w_{n-1} \ldots\ldots w_1$

Substring:
Z is a substring of w if z appears consecutively within w .
e.g " deck" is a substring of abcdeckabcjkl.

Concatenation
Two strings over the same alphabet can be combined to form a third by operation of concatenation. The concatenation of strings x and y written xoy or simply xy is the string x followed by the string y.
Formally w=xy if and only if |w| =|x| + |y|, w(j)= x(j) for j=1 … , |x| and w(|x|+j)=y(j) for j=1 , |y|.
e.g 01o 001=01001
woe=eow =w for any string w . concatenation is associative i.e (wx)y= w(xy) for any string w, x,y.

Suffix:
If w=xy for some x, then y is a suffix of w.
Prefix:
If w=xy for some y, then x is a prefix of w.
Note: for each string w and each natural number i, the string w^i is defined as
$w^0 = e$
$w^{i+1} = w^i w$
$w^1=w$
$do^2 = dodo$

Language

Language L over the alphabet (Σ) is the subset of Σ* . Any set of strings over an alphabet Σ i.e any subset of Σ* will be called a language. Thus Σ* , ø and Σ are languages.

Example

For Σ= { 0, 1}

L= { x : x ϵ { 0,1 } * | x has even number of Zero's }

011011100 ϵ L , 1111 ϵ L , 1011011000 ∉ L

The infinite language L are denoted as

L= { w ϵ Σ * : w has property P}

1.5 Operation on Languages

1. **Concatenation operation**

   If L_1 and L_2 are languages over Σ , their concatenation is L = L1.L2 or L1L2 where

   L= **{** w ϵ Σ * : w = xy for some x ϵ L1 and y ϵ L2 }

   e.g Σ= { 0,1}

   L_1= **{** w ϵ Σ * : w has an even number of 0's}

   L_2= **{** w ϵ Σ * : w starts with a 0 and the rest of the symbols are 1's }

   then

   L_1L_2= **{** w ϵ Σ * : w has an odd number of 0's }
2. **Kleene star**

   Denoted by L* . It is the set of all strings obtained by concatenating zero or more strings from L.

   L*= { w ϵ Σ * : w =w_1o w_2 o ……. o w_k for some K ≥ 0 and some w_1o w_2 o ……. o w_k ϵ L }

   We write L^+ for the language LL*.

   Equivalently L^+ is the language L^+ = { w ϵ Σ * : w =w_1o w_2 o ……. o w_k for some K ≥ 1 and some w_1o w_2 o ……. o w_k ϵ L }
3. **Union**

1.6 A regular expression

A regular expression is a string that describe a whole set of strings, according to certain syntax rules. These expressions are used by many text editor and utilities especially in the unix operation System to search bodies of text for certain patterns and for example, replace the forward strings with a certain other string.

Regular expressions were designed to represent regular languages with a mathematical tool, a tool built from a set of primitives and operations. This representation involves a combination of strings of symbols from some alphabet ∑, parantheses and the operators +, ×, and *.

Formal Definition

The regular expressions over an alphabet $\sum^*$ *are all strings over the alphabet* $\Sigma \cup \{(\ ,\ ), ø, \cup, ^*\}$ *that can be obtained as follows.*

- *ø and each member of Σ is a regular expression.*
- *If α and β are regular expressions, then so is (α β).*
- *If α and β are regular expressions, then so is (α ∪β).*
- *If α is a regular expressions, then so is α* .*
- *Nothing is a regular expression unless it follows from (1) through (4).*

Note: Every regular expression is associated with some language.

Example of regular expression

Assume that Σ = {a, b, c}

- *Zero or more: a** means "zero or more *a*'s",
- To say "zero or more *ab*'s," i.e., {e, *ab, abab,* …..} you need to say (*ab*)*.
- *One or more:* Since *a** means "zero or more *a's*", you can use *aa** (or equivalently *a***a*) to mean "one or more *a*'s". Similarly to describe 'one or more *ab*'s", that is {*ab*, *abab*, *ababab*, KK}, you can use *ab* (*ab*)*.
- *Zero or one:* It can be described as an optional '*a*' with (*a* + e).
- *Any string at all:* To describe any string at all (with Σ= { *a*, *b*, *c*} you can use (*a* + *b* + *c*)*.
- *Any nonempty string:* This is written any character from Σ= {*a*, *b*, *c*} followed by any string at all: (*a* + *b* + *c*) (*a* + *b* + *c*)*
- *Any string not containing* ....: To describe any string at all that does not contain an '*a*' (with Σ={*a*, *b*, *c*}), you can use (*b* + *c*)*.
- *Any string containing exactly one ....:* To describe any string that contains exactly one '*a*' put "any string not containing an *a*", on either side of the '*a*' like: (*b* + *c*)* *a*(*b* + *c*)* .

<u>The Operators of Regular Expressions:</u>

+, **.** , * (Union, concatenation , kleene star)

<u>Precedence of Regular-Expression Operators</u>

- The star Operator is of highest precedence.
- Next in precedence comes the concatenation or dot operator.
- Finally, unions are grouped with their operands.

For example:

The expression 10*+0 is grouped (1(0)*)+0.

Difference between L^* and L^+

L^* denotes Kleene closure and is given by $L^* = \bigcup_{i=0}^{\infty} L^i$
Example : $0^* = \{e, 0, 00, 000, \ldots\}$
Language includes empty words also.

L^+ denotes Positive closure and is given by $L^+ = \bigcup_{i=1}^{\infty} L^i$
$L^+ = LL^*$

Regular Languages
The regular languages are those languages that can be constructed from the "big three" set operations viz., (a) Union (b) Concatenation (c) Kleene star.
A regular language is defined as follows.
Definition**:** Let Σ be an alphabet. The class of "regular languages" over $\sum$ is defined inductively as follows**:**

(a) $\varnothing$ is a regular language
(b) For each $\sigma \in \Sigma$, $\{\sigma\}$ is a regular language
(c) For any natural number $n \geq 2$ if $L_1, L_2, \ldots\ldots L_n$ are regular languages, then so is $L_1 \cup L_2 \cup \ldots \ldots \cup L_n$.
(d) For any natural number $n \geq 2$, if $L_1, L_2, \ldots\ldots L_n$ are regular languages, then so is $L_1 \circ L_2 \circ \ldots \ldots \circ L_n$.
(e) If L is a regular language, then so is L^*.
(f) Nothing else is a regular language unless its construction follows from rules (a) to (e).

Examples:

(i) $\varnothing$ is a regular language (by rule (a))
(ii) $L = \{a, ab\}$ is a language over $\Sigma = \{a, b\}$ because, both $\{a\}$ and $\{b\}$ are regular languages by rule (b). By rule (d) it follows that $\{a\} \circ \{b\} = \{ab\}$ is a regular language. Using rule (c), we see that $\{a\} \cup \{ab\} = L$ is a regular language.
(iii) The language over the alphabet $\{0,1\}$ where strings contain an even number of 0's can be constructed by

$$(1^*((01^*)(01^*))^*)$$

or simply $1^*(01^*\ 01^*)^*$.

Formally, the relation between regular expressions and the languages they represent is established by a function L, such that if a is any regular expression, then L(a) is the language represented by a. That is, L is a function from strings to languages. The function £ is defined as follows.

(1) $\mathcal{L}(\oslash) = \emptyset$, and $\mathcal{L}(a) = \{a\}$ for each $a \in \Sigma$.
(2) If α and β are regular expressions, then $\mathcal{L}((\alpha\beta)) = \mathcal{L}(\alpha)\mathcal{L}(\beta)$.
(3) If α and β are regular expressions, then $\mathcal{L}((\alpha \cup \beta)) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$.
(4) If α is a regular expression, then $\mathcal{L}(\alpha^\star) = \mathcal{L}(\alpha)^*$.

Statement 1 defines $\mathcal{L}(\alpha)$ for each regular expression α that consists of a single symbol; then (2) through (4) define $\mathcal{L}(\alpha)$ for regular expressions of some length in terms of $\mathcal{L}(\alpha')$ for one or two regular expressions α' of smaller length. Thus every regular expression is associated in this way with some language.

Example 1.8.2: What is $\mathcal{L}(((a \cup b)^\star a))$? We have the following.

$$\begin{aligned}
\mathcal{L}(((a\cup b)^\star a)) &= \mathcal{L}((a\cup b)^\star)\mathcal{L}(a) \text{ by}(2) \\
&= \mathcal{L}((a\cup b)^\star)\{a\} \text{ by } (1) \\
&= \mathcal{L}((a\cup b))^*\{a\} \text{ by } (4) \\
&= (\mathcal{L}(a) \cup \mathcal{L}(b))^*\{a\} \text{ by } (3) \\
&= (\{a\} \cup \{b\})^*\{a\} \text{ by } (1) \text{ twice} \\
&= \{a, b\}^*\{a\} \\
&= \{w \in \{a, b\}^* : w \text{ ends with an } a\}
\end{aligned}$$

Languages defined by Regular Expressions

There is a very simple correspondence between regular expressions and the languages they denote:

Regular expression	L (Regular Expression)
x, for each $x \in \Sigma$	$\{x\}$
λ	$\{\lambda\}$
$\varnothing$	$\{\ \}$
(r_1)	$L(r_1)$
r_1^*	$(L(r_1))^*$
$r_1 r_2$	$L(r_1)L(r_2)$
$r_1 + r_2$	$L(r_1) \cup L(r_2)$

Algebraic Laws for RE's

- Commutative law for union: $L + M = M + L$
- Associative law for union: $(L + M) + N = L + (M + N)$
- Associative law for concatenation: $(LM)N = L(MN)$, Note that there is no commutative law for
- Concatenation, i.e. $LM \neq ML$
- The identity for union is: $L + \emptyset = \emptyset + L = L$
- The identity for concatenation is: $L.\ e = e.\ L = L$
- The annihilator for concatenation is: $\emptyset L = L\emptyset = \emptyset$
- Left distributive law: $L(M + N) = LM + LN$
- Right distributive law: $(M + N)L = LM + LN$
- Idempotent law: $L + L = L$

Laws Involving Closure

- $(L^*)^* = L^*$ – i.e. closing an already closed expression does not change the language
- $\emptyset^* = e$
- $e^* = e$
- $L+ = LL^* = L^*L$ – more of a definition than a law
- $L^* = L^+ + e$
- $L? = e + L$ – more of a definition than a law

Checking a Law

Suppose we are told that the law $(R + S)^* = (R^*S^*)^*$ holds for regular expressions. How would we check that this claim is true?

1. Convert the RE's to DFA's and minimize the DFA's to see if they are equivalent
2. We can use the "concretization" test:
   - Think of R and S as if they were single symbols, rather than placeholders for languages, i.e., R = {0} and S = {1}.
   - Test whether the law holds under the concrete symbols. If so, then this is a true law, and if not then the law is false.

Concretization Test

For our example : $(R + S)^* = (R^*S^*)^*$

We can substitute 0 for R and 1 for S. The left side is clearly any sequence of 0's and 1's. The right side also denotes any string of 0's and 1's, since 0 and 1 are each in $L(0^*1^*)$.

• NOTE: extensions of the test beyond regular expressions may fail.
Consider the "law" $L \cap M \cap N = L \cap M$.
This is clearly false
– Let L=M={a} and N= Ø. {a} ≠Ø.
– But if L={a} and M = {b} and N={c} then
– L∩M does equal L∩M ∩N which is empty.
– The test would say this law is true, but it is not because we are applying the test beyond regular expressions.