

## 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 1930s, Gödel, Turing, and Church discovered that some of the fundamental mathematical problems cannot be solved by a computer. (This sounds strange, because computers were invented only in the 1940s).

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 *1/2-hard*, if it cannot be solved efficiently, or if we don't know whether it can be solved efficiently. Examples of *1/2-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 1/2-hard are really 1/2-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,  $b \in L$ . 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  $z \notin L$ .

### **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  $\emptyset$ . Any set other than the empty set is said to be nonempty.

## **1.2 Set Operations**

**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  $B$ .

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

**Complement**

The complement of set  $A$ , written as  $A^c$  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 :**  $(A \cup B) \cup C = A \cup (B \cup C)$

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



### Additional Terminology

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

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

**(c) Powerset.** The powerset of a set, written  $2A$ , 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  $x \in A$  and  $y \in B$  is called the Cartesian Product of the sets  $A$  and  $B$  and is denoted by  $A \times B$ , i.e.  
 $A \times B = \{ (x, y) : x \in A \text{ and } y \in B \}$

### **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)  $s \in S$  ( $s$  is a member of  $S$ )

(b)  $t \in T$

(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.

**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  $R$  on  $A \times A$  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  $R$  on  $A \times A$  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  $(a, b) \in R$  and  $(b, c) \in R$ , then  $(a, c) \in R$ . The relation  $\{(a, b) : a, b \in P \text{ and } a \text{ is an ancestor of } b\}$  is transitive, since if  $a$  is an ancestor of  $b$  and  $b$  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  $A \times A$  is called an equivalence relation on  $A$  if  $R$  satisfies the following conditions:

- (i)  $(a, a) \in R$  for all  $a \in A$  ( $R$  is reflexive)
- (ii) If  $(a, b) \in R$ , then  $(b, a) \in R$ , then  $(a, b) \in R$  ( $R$  is symmetric)
- (iii) If  $(a, b) \in R$  and  $(b, c) \in R$ , then  $(a, c) \in R$  ( $R$  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  $\frac{1}{2}$ Partial ordering or a  $\frac{1}{2}$ Partial ordering 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  $\frac{1}{2}$ Partially ordered set or  $\frac{1}{2}$ Poset.

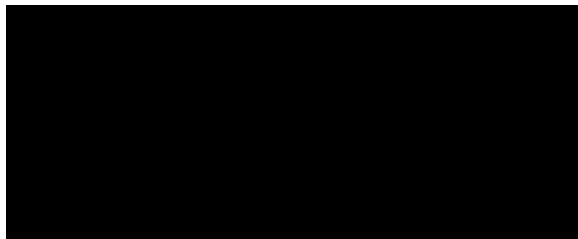
***Partition***

A Partition  $P$  of  $S$  is a collection  $\{A_i\}$  of nonempty subsets of  $S$  with the properties:

- (i) Each  $a \in S$  belongs to some  $A_i$ ,
- (ii) If  $A_i \cap A_j \neq \emptyset$ , then  $A_i = A_j$

***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  $\frac{1}{2}$ function.



A function is otherwise known as  $\frac{1}{2}$ 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  $A \times B$ , 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  $a = a'$ .

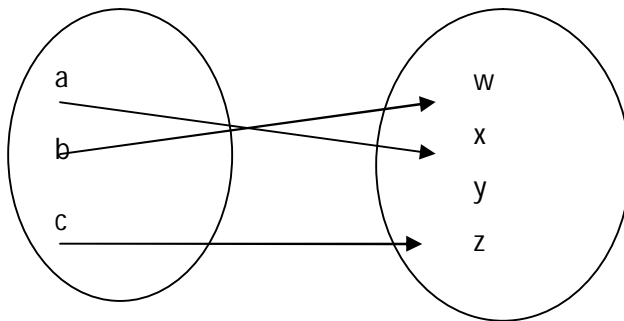


Fig: One to one function (Injection)

(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  $B$ .

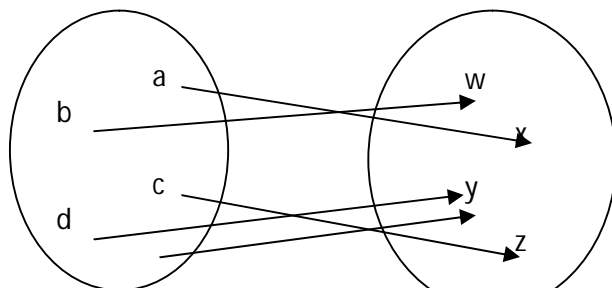


Fig: Surjection

(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.

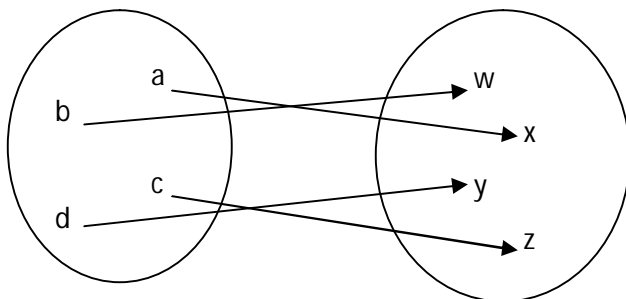


Fig: Bijection

**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

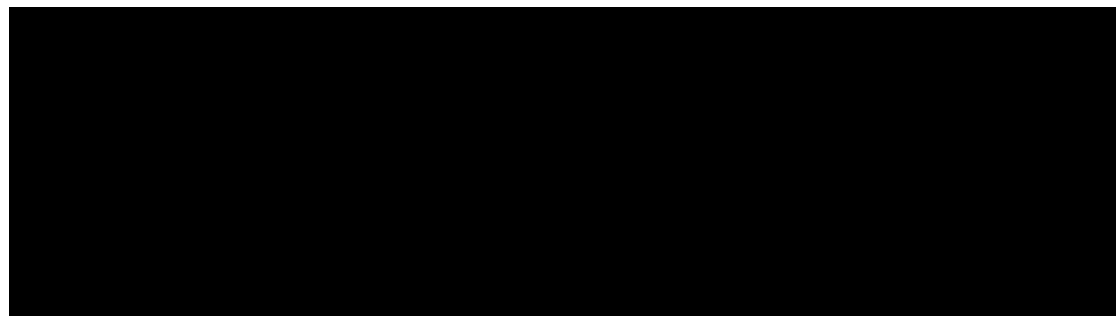
**Example**

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

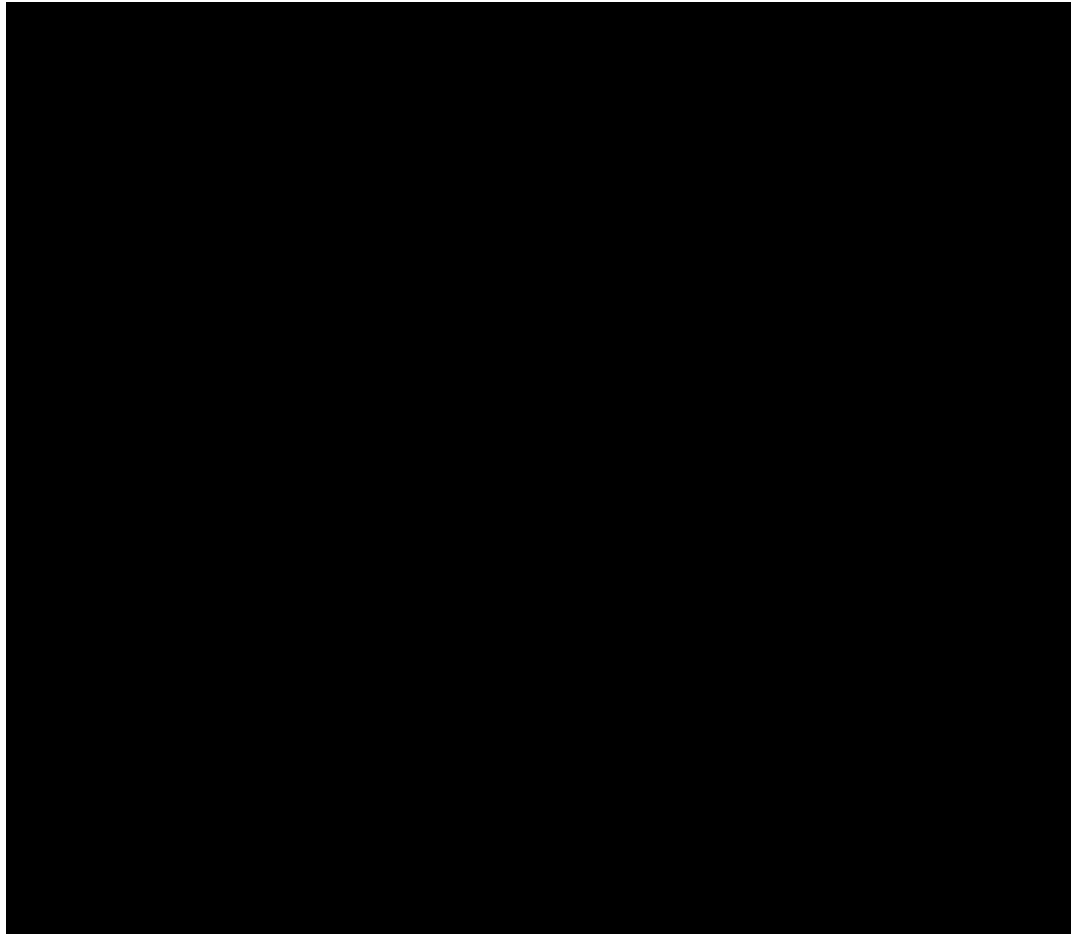
Basic Step. Let  $n=0$ . Then the sum on the left is zero, since there is nothing to add. The expression on the right is also zero.

Induction hypothesis:

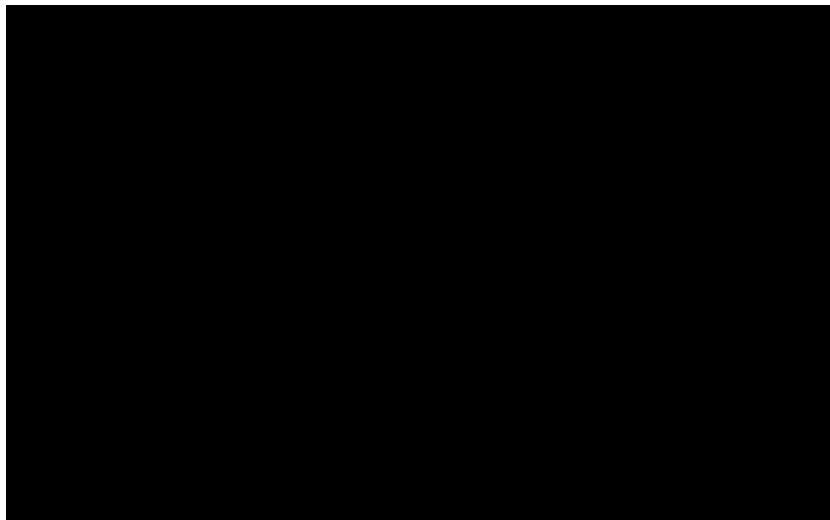
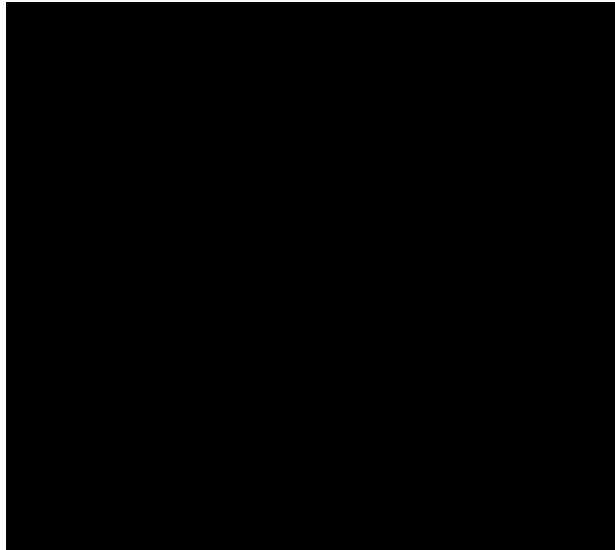
Assume that, for some  $m \geq 0$ ,  $1+2+3+\dots+m = \frac{m(m+1)}{2}$







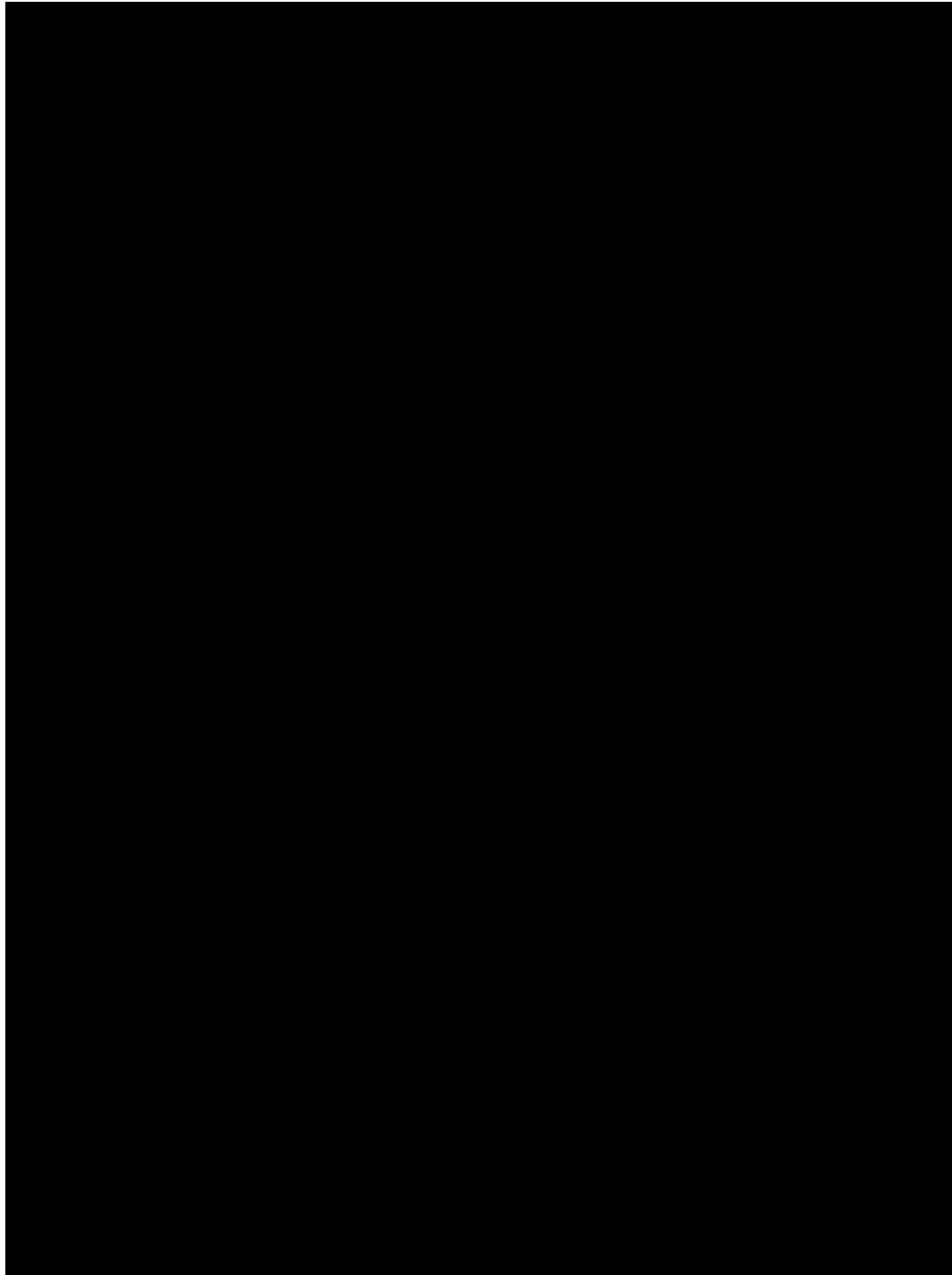


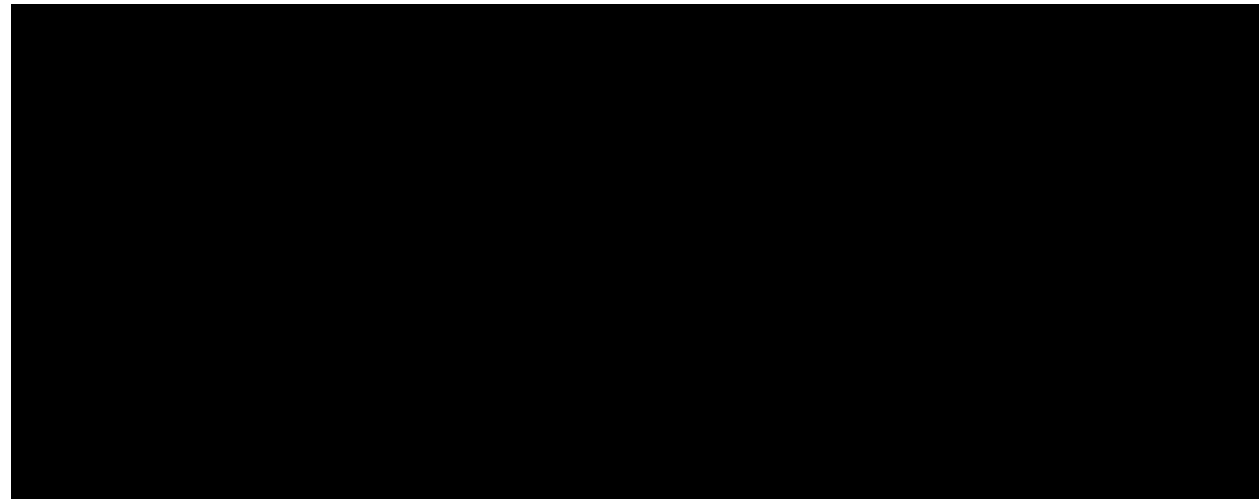




### Diagonalization Principle

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





## 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  $\frac{1}{2}$ pigeons) with elements of  $B$  (the  $\frac{1}{2}$ 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.

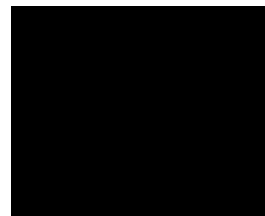
Thus if  $S_1$  and  $S_2$  are two non empty finite sets and  $|S_1| > |S_2|$ , then there is no one-to-one function from  $S_1$  to  $S_2$ . Pigeonhole principle can be used to show that certain languages are not regular. We will use pigeonhole principle in the topic pumping lemma later.

## 1.4 Some Terminologies

### Alphabets :

An alphabet is a finite, nonempty set of symbols. The alphabet of a language is normally denoted by  $\Sigma$ .

Example :



### Strings or Words over Alphabet :

A string or word over an alphabet  $\Sigma$  is a finite sequence of concatenated symbols of  $\Sigma$ . Denoted by  $\Sigma^*$ .

**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  $\epsilon$ .

### Reverse string:

If  $w = w_1w_2\ldots w_n$  where each  $w_i$  is a symbol, the reverse of  $w$  is  $w_nw_{n-1}\ldots w_1$ .

### 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  $j=1, \ldots, |x|$  and  $w(|x|+j)=y(j)$  for  $j=1, \ldots, |y|$ .

e.g.  $010001=01001$

$w\epsilon = \epsilon w = w$  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

$$w^0 = e$$

$$w^{i+1} = w^i w$$

$$w^1 = w$$

$$w^2 = ww$$

**Language**

Language  $L$  over the alphabet  $(\Sigma)$  is the subset of  $\Sigma^*$ . Any set of strings over an alphabet  $\Sigma$  i.e any subset of  $\Sigma^*$  will be called a language. Thus  $\Sigma^*$ ,  $\{0,1\}^*$  and  $\{0,1\}$  are languages.

Example

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

$L = \{x : x \in \{0,1\}^* \mid x \text{ has even number of Zero's}\}$

$011011100 \in L, 1111 \in L, 1011011000 \in L$

The infinite language  $L$  are denoted as

$L = \{w \in \Sigma^* : w \text{ has property } P\}$

**1.5 Operation on Languages****1. Concatenation operation**

If  $L_1$  and  $L_2$  are languages over  $\Sigma$ , their concatenation is  $L = L_1.L_2$  or  $L_1L_2$  where

$L = \{w \in \Sigma^* : w = xy \text{ for some } x \in L_1 \text{ and } y \in L_2\}$

e.g  $\Sigma = \{0,1\}$

$L_1 = \{w \in \Sigma^* : w \text{ has an even number of 0's}\}$

$L_2 = \{w \in \Sigma^* : w \text{ starts with a 0 and the rest of the symbols are 1's}\}$

then

$L_1L_2 = \{w \in \Sigma^* : w \text{ has an odd number of 0's}\}$

**2. Kleene star**

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

$L^* = \{w \in \Sigma^* : w = w_1 \circ w_2 \circ \dots \circ w_K \text{ for some } K \geq 0 \text{ and some } w_1 \in L, w_2 \in L, \dots, w_K \in L\}$

We write  $L^+$  for the language  $LL^*$ .

Equivalently  $L^+$  is the language  $L^+ = \{w \in \Sigma^* : w = w_1 \circ w_2 \circ \dots \circ w_K \text{ for some } K \geq 1 \text{ and some } w_1 \in L, w_2 \in L, \dots, w_K \in L\}$

**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  $\Sigma$ , parantheses and the operators  $+$ ,  $\times$ , and  $*$ .



The regular expressions over an alphabet  $\Sigma$  are all strings over the alphabet  $\Sigma \cup \{ (, ), \dot{\imath};^{1/2}, * \}$  that can be obtained as follows.

If  $r$  and  $s$  are regular expressions, then so is  $(r \cup s)$ .

If  $r$  and  $s$  are regular expressions, then so is  $(r \cup s)$ .

If  $r$  is a regular expressions, then so is  $r^*$ .

*Nothing is a regular expression unless it follows from (1) through (4).*

*Note: Every regular expression is associated with some language.*

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

*Zero or more:*  $a^*$  means  $\text{zero or more } a$

To say  $i; 1/2$  zero or more  $i; 1/2$  is  $ab, abab, i; 1/2..$  you need to say  $(ab)^*$ .

*One or more:* Since  $a^*$  means  $i_{\frac{1}{2}}^1$  zero or more  $i_{\frac{1}{2}}^1$ 's, you can use  $aa^*$  (or equivalently  $a^*a$ ) to mean  $i_{\frac{1}{2}}^1$  one or more  $i_{\frac{1}{2}}^1$ 's. Similarly to describe  $i_{\frac{1}{2}}^1$  one or more  $ab$ 's,  $abab$ ,  $ababab$ ,  $KK$ , you can use  $ab(ab)^*$ .

*Zero or one:* It can be described as an optional  $i\pi/2$  with (+ e).

*Any string at all:* To describe any string at all (with  $\Sigma = \{ a, b, c \}$ ) you can use  $(a + b + c)^*$ .

*Any nonempty string:* This is written any character from  $\Sigma = \{a, b, c\}$  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  $\text{if}^{\frac{1}{2}}$  (with  $\Sigma = \{a, b, c\}$ ), you can use  $(b + c)^*$ .

**Any string containing exactly one  $a$ :** To describe any string that contains exactly one  $a$ , we put  $i$  any string containing an  $a$ , on either side of the  $a$ , like  $(b + c)^* a (b + c)^*$ .

 $+, \cdot, *$  (Union, concatenation, kleene star)

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 :  $0^* = \{ \epsilon, 0, 00, 000, \dots \}$

Language includes empty words also.

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

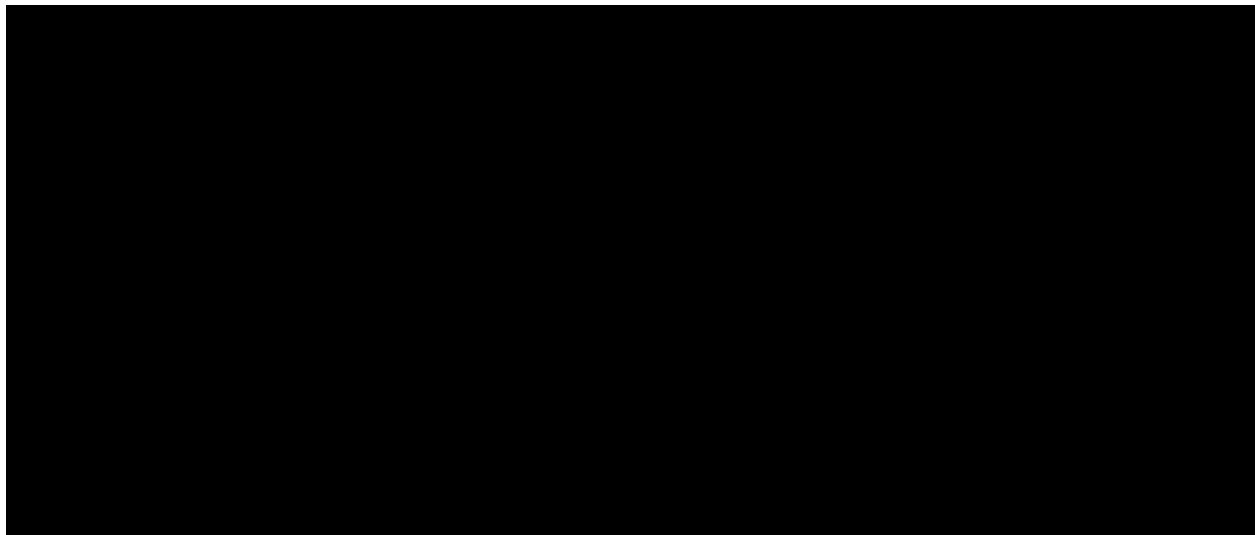
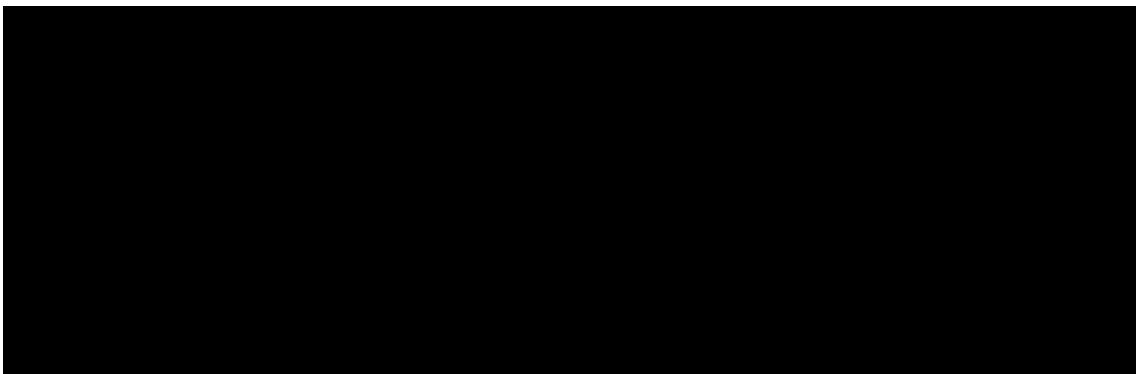
$L^+ = LL^*$

**Regular Languages**

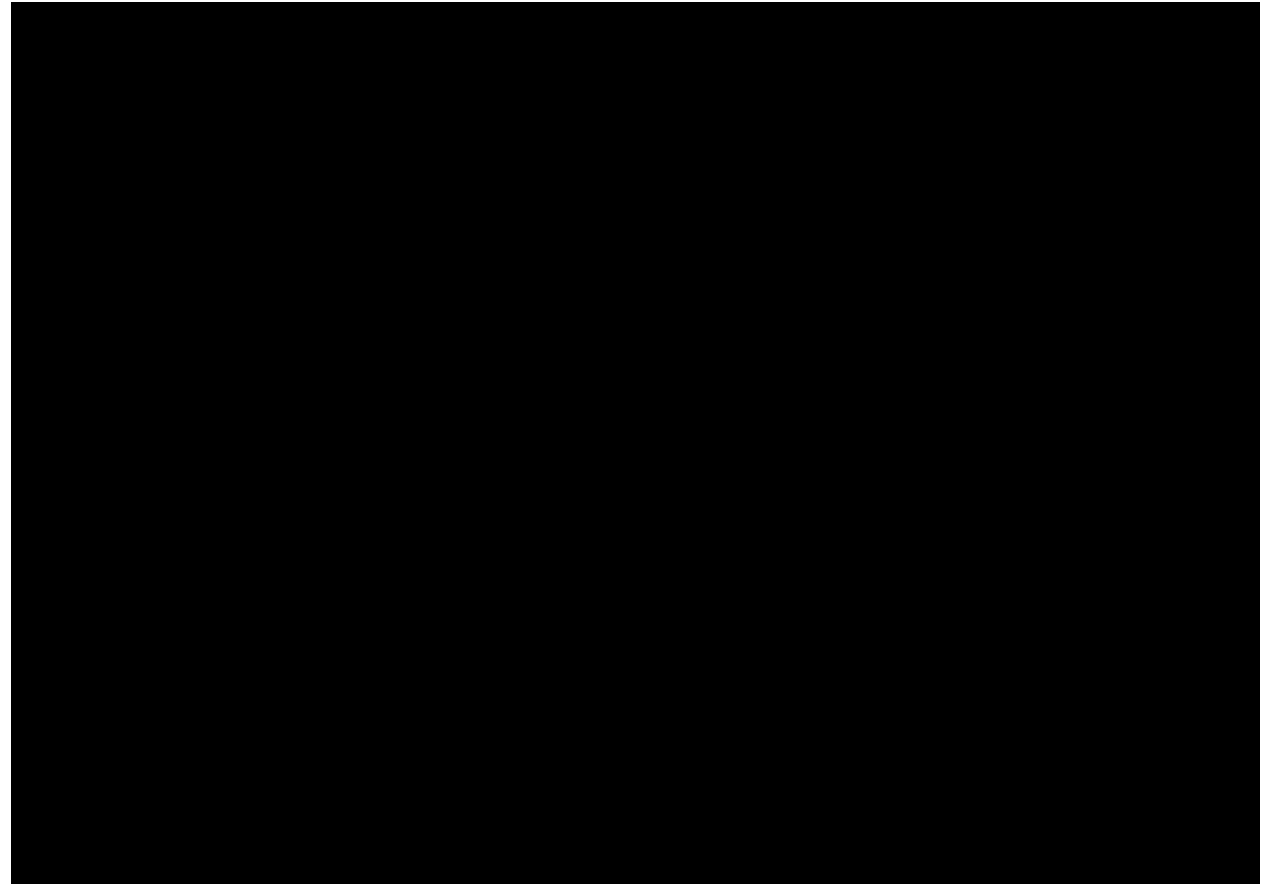
The regular languages are those languages that can be constructed from the three set operations viz., (a) Union (b) Concatenation (c) Kleene star.

A regular language is defined as follows.

**Definition:** Let  $\Sigma$  be an alphabet. The class of regular languages over  $\Sigma$  is defined inductively as follows:

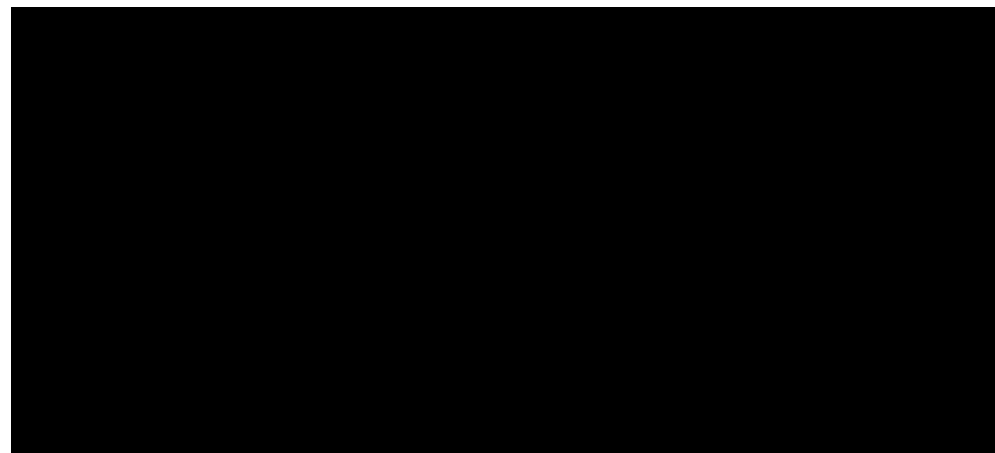


Formally, the relation between regular expressions and the languages they represent is established by a function  $L$ , such that if  $a$  is any regular expression, then  $L(a)$  is the language represented by  $a$ . That is,  $L$  is a function from strings to languages. The function  $L$  is defined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



### Languages defined by Regular Expressions

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



## Algebraic Laws for REs

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:  $L \cdot e = e \cdot L = L$

The annihilator for concatenation is:  $\emptyset \cdot L = L \cdot \emptyset = \emptyset$

Left distributive law:  $L(M + N) = LM + LN$

Right distributive law:  $(M + N)L = ML + NL$

Idempotent law:  $L + L = L$

## Laws Involving Closure

$(L^*)^* = L^*$  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 REs to DFA and minimize the DFA. 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  $L \cap M = L \cup M$ .

This is clearly false

Let  $L=M=\{a\}$  and  $N=\{a\}$ .

But if  $L=\{a\}$  and  $M=\{b\}$  and  $N=\{c\}$  then

$M$  does equal  $L \cap M \cap N$  which is empty.

The test would say this law is true, but it is not because we are applying the test beyond regular expressions.