Mathematical Foundations

Nabendu Chaki
University of Calcutta
nabendu@ieee.org

Introduction

Software professionals live with programs. In a very simple language, one can program only for something that follows some well understood, non-ambiguous logic. The KA on mathematical foundation helps to comprehend this logic that in turn is translated into programming language code. In this KA, mathematics that has been the primary focus is so different from typical arithmetic where numbers are dealt and discussed. The essence of mathematics for a software engineer has to primarily address the issues of logic and reasoning.

Mathematics, in a sense, is the study of formal systems. The word "formal" is associated with preciseness, so that there can not be any ambiguous or erroneous interpretation of the fact. Mathematics is therefore the study of any and all certain truths about any concept. This concept can be about numbers, as well as about symbols, images, sounds, video, almost anything! In short, numbers and numeric equations aren't only subject to preciseness. On the contrary, a software engineer needs to have a precise abstraction on a diverse application domain.

The Knowledge Area on mathematical foundations in SWEBOK covers the basic techniques to identify a set of rules for reasoning in the context of the system under study. Anything that one can deduce following these rules is an absolute certainty within the context of that system. In this KA, techniques have been defined and discussed that can represent and take forward the reasoning and judgment of a software engineer in a precise (and hence, mathematical) manner. The language and methods of logic that have been discussed here allow us to describe mathematical proofs to infer conclusively the absolute truth of certain concepts beyond the numbers. In short, you can write a program for a problem only if it follows some logic. The objective of introducing a separate KA on mathematical foundation is to develop a skill in you to identify and describe such logic. The emphasis is to help you to understand the basic concepts rather than challenging your arithmetic abilities!

1.1. Set, Relations, Functions [Chapter 2, Rosen-2011]

Set: A *set* is a collection of objects, called elements of the set. A set can be represented by listing its elements between braces, e.g., $S = \{1, 2, 3\}$.

The symbol $\in$ is used to express that an element belongs to a set, or in other words is a member of the set. Its negation is represented by $\notin$, e.g., $1 \in S$, but $4 \notin S$.

In a more compact representation of set using set builder notation $\{x|P(x)\}$ is the set of all x such that P(x) for any proposition P(x) over any universe of discourse. Examples for some important sets include the following:

$N = \{0, 1, 2, 3, \cdots\}$ = the set of non-negative integers.

$Z = \{\cdots, -3, -2, -1, 0, 1, 2, 3, \cdots\}$ = the set of integers.

Finite and infinite Set: A set with a finite number of elements is called a *finite set*. Conversely, any set that does not have a finite number of elements in it is an *infinite set*. The set of all natural numbers, as for example, is an infinite set.

Cardinality: The *cardinality* of a finite set S, is the number of elements in S. This is represented $|S|$, e.g. if $S = \{1, 2, 3\}$ then $|S| = 3$.

Universal set: In general $S = \{x \in U \mid p(x)\}$, where U is the universe of discourse in which the predicate P(x) must be interpreted. The "universe of discourse" for a given predicate is often referred as *universal set*. Alternately one may define *universal set* as the set of all elements.

Set Equality: Two sets are *equal* if and only if they have the same elements, i.e.:

$X = Y \equiv \forall p\, (p\in X \leftrightarrow p \in Y)$.

Subset: X is a *subset* of set Y, or X is contained in Y if all elements of X are included in Y. This is denoted by $X \subseteq Y$. In other words, $X \subseteq Y$ iff $\forall p(p\in X \rightarrow p \in Y)$.

e.g., if $X = \{1, 2, 3\}$ and $Y = \{1, 2, 3, 4, 5\}$ then $X \subseteq Y$.

If X is not a subset of Y, it is denoted as $X \subsetneq Y$.

Proper subset: X is a *proper subset* of Y (denoted by if $X \subset Y$) if X is a subset of Y, but X is not equal to Y, i.e., there is some element in Y that is not in X.

In other words, $X \subset Y$ if $(X \subseteq Y) \wedge (X \neq Y)$.

e.g., if $X = \{1, 2, 3\}$, $Y = \{1, 2, 3, 4\}$ and $Z=\{1, 2, 3\}$ then $X \subset Y$, but X is not a proper subset of Z. Sets X and Z are equal sets.

If X is not a proper subset of Y, it is denoted as $X \not\subset Y$.

Superset: If X is a subset of Y, then Y is called a *superset* of X. This is denoted by $Y \supseteq X$, i.e., $Y \supseteq X$ iff $X \subseteq Y$.

e.g., if $X = \{1, 2, 3\}$ and $Y = \{1, 2, 3, 4, 5\}$ then $Y \supseteq X$.

Empty Set: A set with no elements is called *empty set*. An empty set, denoted by ϕ, is also referred as null or void set.

Power Set: The set of all subsets of a set X is called the *power set* of X. It is represented P(X),

e.g., if $X = \{a, b, c\}$, then $P(X) = \{\phi, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$. If $|X| = n$ then $|P(X)| = 2^n$

Venn Diagrams: Venn diagrams are graphic representations of sets as enclosed areas in the plane.

e.g., in figure 1, the rectangle represents the universal set and the shaded region represents a set X.

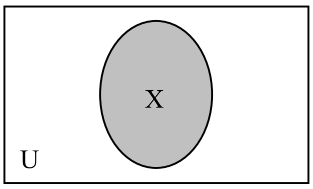

Fig. 1: Venn Diagram for set X

Set Operations

1. Intersection: The *intersection* of two sets X and Y, denoted by X ∩ Y, is the set of common elements in both X and Y.

i.e., $X \cap Y = \{p \mid (p \in X) \wedge (p \in Y)\}$.

As for example, {1, 2, 3} ∩ {3, 4, 6} = {3}

If X ∩ Y = ϕ, then the two sets X and Y are said to be disjoint pair of sets.

A Venn diagram for set intersection is shown in figure 2. The common portion of the two sets marked with overlapping strips represents set intersection.

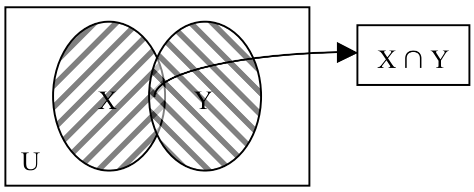

Fig. 2: Intersection of Set X and Y

2. Union: The *union* of two sets X and Y, denoted by X ∪ Y, is the set of all elements either in X or in Y, or in both.

i.e., $X \cup Y = \{p \mid (p \in X) \vee (p \in Y)\}$.

As for example, {1, 2, 3} ∪ {3, 4, 6} = {1, 2, 3, 4, 6}

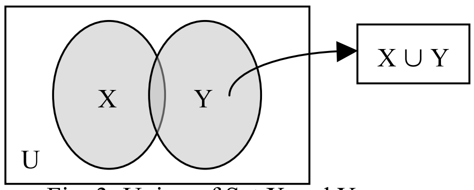

Fig. 3: Union of Set X and Y

It may be noted that |X∪Y| = |X| + |Y| – |X∩Y|.

A Venn diagram illustrating union of two sets is represented by the shaded region in figure 3.

3. Complement: The set of elements in the universal set that do not belong to a given set X is called its *complement set* X'.

i.e., $X' = \{p \mid (p \in U) \wedge (p \notin X)\}$.

The shaded portion of the Venn diagram in figure 4 represents the complement set of X.

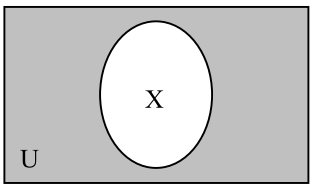

Fig. 4: Venn Diagram for complement set of X

4. Difference or Relative Complement: The set of elements that belong to a set X but not to another Y builds the *difference* of Y from X. This is represented by X - Y.

i.e., $X - Y = \{p \mid (p \in X) \wedge (p \notin Y)\}$.

As for example, {1,2,3} - {3,4,6} = {1,2}

It may be proved that X - Y = X ∩ Y'.

Set difference X – Y is illustrated by the shaded region in figure 5 using Venn diagram.

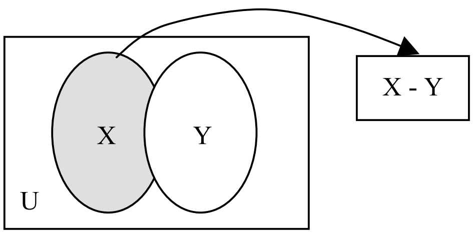

Fig. 5: Venn Diagram for X -Y

Cartesian Product: An ordinary pair{p, q} is a set with two elements. In a set the order of the elements is irrelevant, so {p, q} = {q, p}.

In an ordered pair (p, q), the order of occurrences of the elements is relevant. Thus, (p, q) ≠ (p, q) unless p = q. In general (p, q) = (s, t) iff p = s and q = t.

Given two sets X and Y, their *cartesian product* X x Y is the set of all ordered pairs (p, q) such that p ∈ X and q ∈ Y.

i.e., $X \times Y = \{(p, q) \mid (p \in X) \wedge (q \in Y)\}$.

As for example, {a,b} x{1,2} = {(a,1), (a,2), (b,1), (b,2)}

Properties of Set

Some of the important properties and laws of sets are mentioned below.

1. Associative Laws:

X ∪ (Y ∪ Z) = (X ∪ Y) ∪ Z

X ∩ (Y ∩ Z) = (X ∩ Y) ∩ Z

2. Commutative Laws:

X ∪ Y = Y ∪ X   X ∩ Y = Y ∩ X

3. Distributive Laws:

X ∪ (Y ∩ Z) = (X ∪ Y) ∩ (X ∪ Z)

X ∩ (Y ∪ Z) = (X ∩ Y) ∪ (X ∩ Z)

4. Identity Laws:

X ∪ ϕ = X   X ∩ U = X

5. Complement Laws:

X ∪ X' = U   X ∩ X' = ϕ

6. Idempotent Laws:

X ∪ X = X   X ∩ X = X

7. Bound Laws:

X ∪ U = U   X ∩ ϕ = ϕ

8. Absorption Laws:

X ∪ (X ∩ Y) = X   X ∩ (X ∪ Y) = X

9. DeMorgan's Laws:

(X ∪ Y)' = X' ∩ Y'   (X ∩ Y)' = X' ∪ Y'

Relation and Function:

A *relation* is an association between two sets of information. As for example, let's consider a set of residents of a city and their phone numbers. The pairing of names with corresponding phone numbers is a relation. This pairing is *ordered* for the entire relation. In the example being considered, for each pair, either the name comes first followed by the phone number or the reverse. The set from which the first element is drawn is called the *domain* set and the other set is called the *range set*. The domain is what you start with and the range is what you end up with.

A *function* is a *well-behaved* relation. A relation R(X, Y) is well behaved if the function maps every element of the domain set X to a single element of the range set Y. A person may have more than one phone numbers in the example being considered. Thus this relation is not a function. However, if we draw a relation between names of residents and their date of births with the name set as domain, then this becomes a well-behaved relation and hence a function. This means that, while all functions are relations, not all relations are functions. In case of a function given an x, one gets one and exactly one y for each ordered pair (x, y).

1.2 Basic Logic [Chapter 1, Rosen-2011]

1.2.1 Propostional Logic: A *proposition* is a statement that is either true or false, but not both. Let's consider declarative sentences for which it is meaningful to assign either of the two status values *true* or *false*. Some examples of proposition are given below.

1. Sun is a star

2. Elephants are mammals.

3. 2+3 = 5, etc.

However, a+3 = b is not a proposition as it is neither true nor false. It depends on the values of the variables *a* and *b*.

The Law of Excluded Middle: For every proposition p, either p is true or p is false.

The Law of Contradiction: For every proposition p, it is not the case that p is both true and false.

Propositional logic is the area of logic that deals with propositions. A *truth table* displays the relationships between the truth values of propositions.

A *Boolean variable* is one whose value is either true or false. Computer bit operations correspond to logical operations of Boolean variables.

The basic logical operators including negation ($\neg p$), conjunction ($p \wedge q$), disjunction ($p \vee q$), exclusive or ($p \oplus q$), implication ($p \rightarrow q$), are to be studied. Compound propositions may be formed using various logical operators.

A compound proposition that is always true is a *tautology*. A compound proposition that is always false is a *contradiction*. A compound proposition that is neither a tautology nor a contradiction is a *contingency*.

Compound propositions that always have the same truth value are called *logically equivalent* (denoted by ≡). Some of the common equivalences are:

Identity laws:

$p \wedge T \equiv p$ $\qquad$ $p \vee F \equiv p$

Domination laws:

$p \vee T \equiv T$ $\qquad$ $p \wedge F \equiv F$

Idempotent laws:

$p \vee p \equiv p$ $\qquad$ $p \wedge p \equiv p$

Double negation law:

$\neg(\neg p) \equiv p$

Commutative laws:

$p \vee q \equiv q \vee p$ $\qquad$ $p \wedge q \equiv q \wedge p$

Associative laws:

$(p \vee q) \vee r \equiv p \vee (q \vee r)$ $\quad$ $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$

Distributive laws:

$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$

De Morgan's laws:

$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\qquad$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$

1.2.2 Predicate logic: A predicate is a verb phrase template that describes a property of objects, or a relationship among objects represented by the variables, e.g., in the sentence *The flower is red* the template *is red* is a predicate. It describes the property of a flower. The same predicate may be used in other sentences too.

Predicates are often given a name, e.g., "Red" or simply "R" can be used to represent the predicate *is red*. Assuming R as the name for the predicate *is red*, sentences that assert an object is of red color can be represented as *R(x)*, where *x* represents an arbitrary object. *R(x)* reads as *x is red*.

Quantifiers allow statements about entire collections of objects rather than having to enumerate the objects by name.

The Universal quantifier ∀x asserts that a sentence is true for all values of variable x.

e.g., $\forall x$ Tiger(x) → Mammal(x) The expression means all tigers are mammals.

The Existential quantifier ∃x asserts that a sentence is true for at least one value of a variable x

e.g., $\exists x$ Tiger(x) → Man-eater(x) The expression means there exists at least one tiger that is man-eater.

Thus, while universal quantification uses implication, the existential quantification naturally uses conjunction.

A variable x that is introduced into a logical expression by a quantifier is bound to the closest enclosing quantifier.

A variable is said to be a free variable if it is not bound to a quantifier.

Similar to that in a block structured programming language, a variable in a logical expression refers to the closest quantifier within whose scope it appears.

e.g., $\exists x$ (Cat(x) $\wedge$ $\forall x$ (Black (x)))

Here, x in Black(x) is universally quantified. The expression implies that cats exist and everything is black.

Propositional logic falls short to represent many assertions that are used in computer science and mathematics. It also fails to compare equivalence and some other types of relationship between propositions.

As for example, the assertion *a is greater than 1* is not a proposition because one can not infer whether it is true or false without knowing the value of *a*. Thus the propositional logic can not deal with such sentences. However, such assertions appear quite often in mathematics and we want to infer on those assertions. Also the pattern involved in the following two logical equivalences can not be captured by the propositional logic: "*Not all men is smoker*" and "*Some men don't smoke*". Each of these two propositions is treated independently in propositional logic. There is no mechanism in propositional logic to find out whether or not the two are equivalent to one another. Hence, in propositional logic, each of the equivalent propositions is treated individually rather than dealing with a general formula that covers all these equivalences collectively.

The predicate logic is supposed to be a more powerful logic that addresses these issues. In a sense, Predicate logic (also known as first-order logic or predicate calculus) is an extension of propositional logic to formulas involving terms and predicates.

1.3 Proof Techniques [Chapter 1, Rosen-2011]

A *proof* is an argument that rigorously establishes the truth of a statement. Proofs can themselves be represented formally as discrete structures

Statements used in a proof include *axioms* and *postulates* that are essentially the underlying assumptions about mathematical structures, the hypotheses of the theorem to be proved, and previously proved theorems.

A *theorem* is a statement that can be shown to be true.

A *lemma* is a simple theorem used in the proof of other theorems.

A *corollary* is a proposition that can be established directly from a theorem that has been proved.

A *conjecture* is a statement whose truth value is unknown.

When a proof of a conjecture is found, the conjecture becomes a theorem. Many times conjectures are shown to be false and hence those are not theorems.

1.3.1 Methods of Proving Theorems

Direct Proof: Direct Proof is a technique to establish that the implication $p \rightarrow q$ is true by showing that q must be true when p is true.

e.g., Show that if n is odd, then $n^2 -1$ is even.

Suppose n is odd. i.e., n = 2k+1, for some integer k.

$\therefore n^2 = (2k+1)^2 = 4k^2+4k+1$.

As the first two terms of the RHS is even number irrespective of the value of k, the LHS, i.e., n^2 is an odd number. Therefore, n^2-1 is even.

Proof by Contradiction: A proposition *p* is true by contradiction is proved based on the truth of the implication $\neg p \rightarrow q$ where q is a contradiction.

e.g., Show that the sum of 2x+1 and 2y−1 is even.

Assume that the sum of 2x+1 and 2y−1 is odd, i.e., 2(x+y) which is a multiple of 2, is odd. This is a contradiction. Hence, the sum of 2x+1 and 2y−1 is even.

An inference rule is a pattern establishing that if a set of premises are all true, then it can be deduced that a certain conclusion statement is true. The reference rules of addition, simplification, and conjunction needs to be studied.

1.4 Basics of Counting [Chapter 6, Rosen-2011]

The *sum rule* states that if a task t_1 can be done in n_1 ways and a second task t_2 can be done in n_2 ways, and if these tasks cannot be done at the same time, then there are $n_1 + n_2$ ways to do either task.

- If A and B are disjoint sets then $|A \cup B|=|A|+|B|$
- In general if A1, A2 . . .An are disjoint sets, then $|A1 \cup A2 \cup \ldots \cup An| = |A1| + |A2| + \ldots + |An|$

e.g., if there are 200 athletes doing sprint events and 30 athletes who participate in long jump event, then how many ways are there to pick one athlete who is either a sprinter or a long jumper?

Using the sum rule, the answer would be 200+30=230.

The *product rule* states that if a task t_1 can be done in n_1 ways and a second task t_2 can be done in n_2 ways after the first task has been done, then there are $n_1 * n_2$ ways to do the procedure.

- If A and B are disjoint sets then $|A \times B|=|A|*|B|$
- In general if A1, A2 . . .An are disjoint sets, then $|A1 \times A2 \times \ldots \times An| = |A1| * |A2| * \ldots * |An|$

e.g., if there are 200 athletes doing sprint events and 30 athletes who participate in long jump event, then how many ways are there to pick two athletes so that one is a sprinter and the other one is a long jumper?

Using the product rule, the answer would be 200*30=6000.

The *principle of inclusion-exclusion* states that if a task t_1 can be done in n_1 ways and a second task t_2 can be done in n_2 ways at the same time with t_1, then to find the total number of ways the two tasks can be done, subtract the number of ways to do both tasks from n_1+n_2.

- If A and B are not disjoint $|A \cup B|=|A| + |B| - |A \cap B|$

In other words, the *principle of inclusion-exclusion* aims to ensure that the objects in the intersection of two sets are not counted more than once!

Recursion is the general term for the practice of defining an object in terms of itself. There are recursive algorithms, recursively defined functions, relations, sets, etc.

A recursive function is a function that calls itself. e.g., we define $f(n)=3*f(n-1)$ for all $n \in N$ and $n \neq 0$ and $f(0) = 5$.

An algorithm is *recursive* if it solves a problem by reducing it to an instance of the same problem with a smaller input.

A phenomenon is said to be *random* if individual outcomes are uncertain but the long-term pattern of many individual outcomes is predictable.

The *probability* of any outcome for a random phenomenon is the proportion of times the outcome would occur in a very long series of repetitions.

The probability P(A) of any event A satisfies 0 ≤ P(A) ≤ 1. Any probability is a number between 0 and 1. If S is the sample space in a probability model, the P(S) = 1. All possible outcomes together must have probability of 1.

Two events A and B are disjoint if they have no outcomes in common and so can never occur together. If A and B are two disjoint events, P(A or B) = P(A) + P(B). This is known as addition rule for disjoint events.

If two events have no outcomes in common, the probability that one or the other occurs is the sum of their individual probabilities.

Permutation is an arrangement of objects in which the order matters without repetition. One can choose r objects in a particular order from a total of n objects by using ^nP_r ways, where, ^np_r = n!/(n-r)!. Various notations like ^nP_r and P(n, r) are used to represent the number of permutations of a set of n objects taken r at a time.

Combination is a selection of objects in which the order does not matter without repetition. This is different from a permutation because the order does not matter. If the order is only changed (and not the members) then no new combination is formed. One can choose r objects in any order from a total of n objects by using ^nC_r ways, where, ^nC_r = n!/[r!*(n-r)!].

1.5 Graphs and Trees [Chapter 10, Chapter 11, Rosen-2011]

1.5.1 Graphs: A graph G = (V, E) where, V is the set of vertices (nodes) and E is the set of edges. Edges are also referred as arcs or links.

F is a function that maps the set of edges E to set of ordered or unordered pairs of elements V.

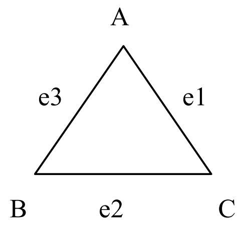

Fig. 6: Example of a Graph

e.g., in figure 6, G = (V, E) where V={A, B, C}, E={e1, e2, e3} and F = {(e1, (A, C)), (e2, (C, B)), (e3, (B, A))}.

The graph in figure 6 is a *simple graph* that consists of a set of vertices or nodes and a set of edges connecting unordered pairs.

The edges in simple graphs are undirected. Such a graph is also referred as *undirected* graphs.

e.g., in figure 6, (e1, (A, C)) may be replaced by (e1, (C, A)) as the pair between vertices A and C is unordered. This holds good for the other two edges too.

In a *multi-graph*, more than one edge may connect the same two vertices. Two or more connecting edges between the same pair of vertices may reflect multiple associations between the same two vertices. Such edges are called parallel or multiple edges.

e.g., in figure 7, the edges e3 and e4 are both between A and B. Figure 7 is a multi-graph where edges e3 and e4 are multiple edges.

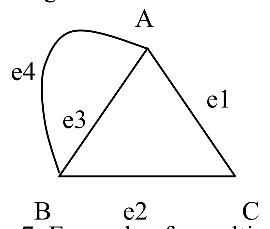

Fig. 7: Example of a multi-graph

In a *pseudo-graph*, edges connecting a node to itself are allowed. Such edges are called loops.

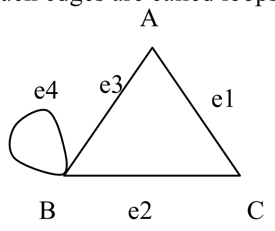

Fig. 8: Example of a pseudo-graph

e.g., in figure 8, the edge e4 both starts and ends at B. Figure 8 is a pseudo graph in which e4 is a loop.

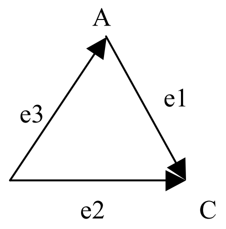

Fig. 9: Example of a directed-graph

A *directed graph* G=(V,E) consists of a set of vertices V and a set of edges E that are ordered pairs of elements of V. A directed graph may contain loops.

e.g., in figure 9, G = (V, E) where V={A, B, C}, E={e1, e2, e3} and F = {(e1, (A, C)), (e2, (B, C)), (e3, (B, A))}.

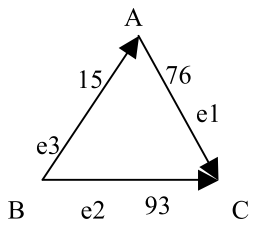

Fig. 10: Example of a weighted graph

In a *weighted graph* G=(V, E) each edge has a weight associated with it. The weight of an edge typically represents the numeric value associated with the relationship between the corresponding two vertices.

e.g., in figure 10, the weights for the edges e1, e2 and e3 are taken to be 76, 93 and 15 respectively. If the vertices A, B and C represent three cities in a state, the weights, for example, could be the distances in miles between these cities.

Let G = (V, E) be an undirected graph with edge set E. Then for an edge e∈E where e = {u, v} the following terminologies are often used:

- u, v are said to be *adjacent* or *neighbors* or *connected*.
- edge e is *incident* with vertices u and v.
- edge e *connects* u and v.
- vertices u and v are *endpoints* for edge e.

If vertex v □ V, the set of vertices in the undirected graph G (V, E) then:

- the *degree* of v, deg(v) is its number of incident edges except that for any self-loops loops are counted twice.
- a vertex with degree 0 is called an *isolated vertex*.
- a vertex of degree 1 is called a *pendant vertex*.

Let G (V, E) be a directed graph. If e(u, v) be an edge of G then the following terminologies are often used:

- u is *adjacent to* v, and v is *adjacent from* u
- e *comes from* u, and *goes to* v.
- e *connects* u to v, or e *goes from* u to v
- the *initial vertex* of e is u
- the *terminal vertex* of e is v

If vertex v is in the set of vertices for the directed graph G (V, E) then:

- *in-degree* of v, $deg^{-}(v)$, is the number of edges going to v, i.e., for which v is terminal vertex.
- *out-degree* of v, $deg^{+}(v)$, is the number of edges coming from v, i.e., for which v is initial vertex.
- *degree* of v, $deg(v) = deg^{-}(v) + deg^{+}(v)$ is the sum of v's in-degree and out-degree.
- a loop at a vertex contributes 1 to both in-degree and out-degree of this vertex

It may be noted that following the definitions above, the degree of a node is unchanged whether we consider its edges to be directed or undirected.

In an undirected graph, a *path* of length n from u to v is a sequence of n adjacent edges from vertex u to vertex v.

• A path is a *circuit* if u=v.

• A path traverses the vertices along it.

• A path is *simple* if it contains no edge more than once.

A *cycle* on n vertices C_n for any n≥3, is a simple graph where $V=\{v_1, v_2, \ldots, v_n\}$ and $E=\{\{v_1,v_2\}, \{v_2,v_3\}, \ldots, \{v_{n-1},v_n\},\{v_n,v_1\}\}$.

e.g., figure 11 illustrates two cycles of length 3 and 4.

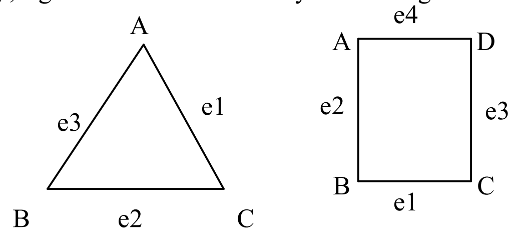

Fig. 11: Example of cycles C_3 and C_4

Adjacency list is a table with one row per vertex, listing its adjacent vertices. The adjacency listing for a directed graph maintains listing of the terminal nodes for vertex in the graph.

Vertex	Adjacency list
A	B, C
B	A, B, C
C	A, B

Vertex	Terminal vertex
A	C
B	A, C
C	-

Fig. 12: Adjacency lists for graphs in figure 8 and figure 9

e.g., figure 12 illustrates the adjacency lists for the pseudo-graph in figure 8 and the directed graph in figure 9. As the out-degree of vertex C in figure 9 is zero, there is no entry against C in the adjacency list.

Different representation for graph like adjacency matrix, incidence matrix, adjacency list needs to be studied.

1.5.1 Trees: A tree T (N, E) is a hierarchical data structure of n = |N| nodes with a specially designated root node R while the remaining n-1 nodes form sub-trees under the root node R. The number of edges |E| in a tree would always be equal to |N|-1.

The *sub-tree* at node X is the sub-graph of the tree consisting of node X and its descendants and all edges incident to those descendants. As an alternate to this recursive definition, a tree may be defined as a connected undirected graph with no simple circuits.

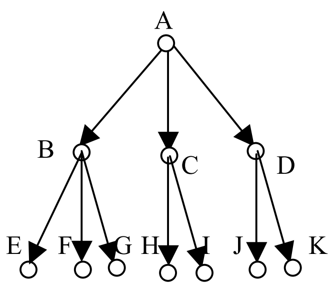

Fig. 13: Example of a Tree

However, one should remember that a tree is strictly hierarchical in nature as compared to a graph which is flat. In case of a tree, an ordered pair is built between two nodes as parent and child. Each child node in a tree is associated with only one parent node, whereas this restriction becomes meaningless for a graph where no parent-child association exists.

An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

Figure 13 presents a tree T(N, E), where the set of nodes N={A, B, C, D, E, F, G, H, I, J, K}. The edge set E is{(A, B), (A, C), (A, D), (B, E), (B, F), (B, G), (C, H), (C, I), (D, J), (D, K)}.

The *parent* of a non-root node *v* is the unique node *u* with a directed edge from u to v. Each node in the tree has a unique parent node except the root of the tree.

e.g., in figure 13, root node A is the parent node for nodes B, C, and D. Similarly B is the parent of E, F and G and so on. The root node A does not have any parent.

A node that has children is called an *internal* node.

e.g., in figure 13, node A or node B are examples of internal nodes.

The *degree* of a node in tree is same as its number of children.

e.g., in figure 13, root node A and its child B both are of degree 3. Nodes C and D have degree 2.

The distance of a node from the root node in terms of number of hops is called its *level*. Nodes in a tree are at different levels. The root node is at level 0. Alternately, the level of a node X is the length of the unique path from the root of the tree to node X.

e.g., root node A is at level 0 in figure 13. Nodes B, C and D are at level 1. The remaining nodes in figure 13 are all at level 2.

The height of a tree is the maximum of the levels of nodes in the tree plus one.

e.g., in figure 13, height of the tree is 2.

A node is called a *leaf* if it has no children. Degree of a leaf node is 0.

e.g., in figure 13, nodes E through K are all leaf nodes with degree 0.

The *ancestors or predecessors* of a non-root node X are all the nodes in the path from root to node X.

e.g., in figure 13, nodes A and D form the set of ancestors for J.

The *successors or descendents* of a node X are all the nodes that has X as its ancestor. For a tree with n nodes, all the remaining n-1 nodes are successors of the root node.

e.g., in figure 13, node B has successors in E, F and G.

If node X is an ancestor of node Y, then node Y is a successor of X.

Two or more nodes sharing the same parent node are called *sibling* nodes.

e.g., in figure 13, node E and G are siblings. However, nodes E and node J, although are from the same level are not sibling nodes.

Two sibling nodes are of same level, but two nodes in the same level are not necessarily siblings.

A tree is called *ordered tree* if the relative position of occurrences of children nodes are significant.

e.g., in a family tree is an ordered tree if as a rule, the name of an elder sibling appears always before (i.e., on the left) the younger sibling.

In an *unordered tree* the relative position of occurrences between the siblings does not bear any significance and may be altered arbitrarily.

A *binary tree* is formed with zero or more nodes where there is a root node R and all the remaining nodes form a pair of *ordered* sub-trees under the root node.

In a binary tree no internal node can have more than 2 children. However, one must consider that besides this criterion in terms of the degree of internal nodes, a binary tree is always ordered. If the positions of the left and right sub-trees for any node in the tree are swapped, then a new tree is derived.

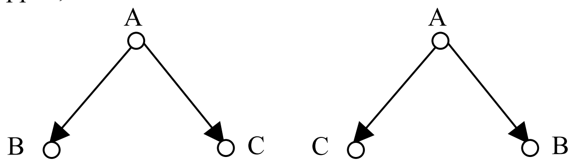

Fig. 14: Examples of Binary Trees

e.g., in figure 14, the two binary trees are different as the positions of occurrences of the children of A are different in the two trees.

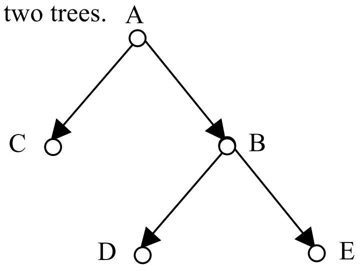

Fig. 15: Example of a Full Binary Tree

According to ***(Rosen 2011),*** a binary tree is called a full binary tree if every internal node has exactly 2 children.

e.g., the binary tree in figure 15, is a full binary tree as both of the two internal nodes A and B are of degree 2.

A full binary tree following the definition above is also referred as a *strictly binary tree*.

A *complete binary tree* has all its levels, except possibly the last one, filled up to the capacity. In case, the last level of a complete binary tree is not full, nodes occur from the leftmost positions available.

e.g., both the binary trees in figure 16 are complete binary trees. The tree in figure 16(a) is a complete as well as full binary tree.

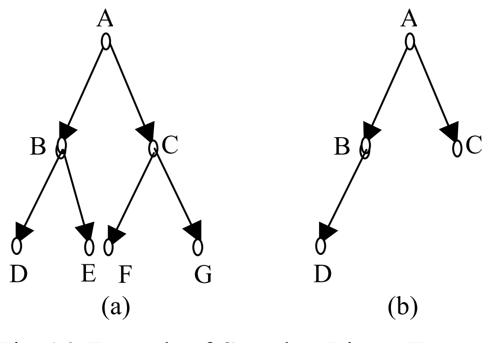

Fig. 16: Example of Complete Binary Trees

Interestingly, following the definitions above, the tree in figure 16(b) is complete but not a full binary tree as node B has only one child in D. On the contrary, the tree in figure 15 is full but not a complete binary tree as the children of B occurs in the tree, where as that for node C does not appear in the last level.

A binary tree of height H is balanced if all its leaf nodes occur at levels H or H-1.

e.g., all the three binary trees in figure 15 and in figure 16 are balanced binary trees.

There are at most 2^H leaves in a binary tree of height H. In other words, if a binary tree with L leaves is full and balanced, then its height is $H = \lceil \log_2 L \rceil$.

e.g., the statement is found true for the two trees in figure 15 and in figure 16(a) as both these trees are full and balanced. However, the expression above does not match for the tree in figure 16(b) as the same is not a full binary tree.

A *binary search tree (BST)* special kind of binary tree in which each node contains a distinct key value, and the key value of each node in the tree is less than every key value in its right sub-tree, and greater than every key value in its left sub-tree.

A traversal algorithm is a procedure for systematically visiting every node of a binary tree. Tree traversals may be defined recursively.

Pre-Order traversal: If T is binary tree with root R and the remaining nodes form an ordered pair of non-null left sub-tree T_L and non-null right sub-tree T_R below R, then the pre-order traversal function PreOrder(T) is defined as:

PreOrder(T) = R, PreOrder(T_L), PreOrder(T_R) … eqn. 1

The recursive process of finding pre-order traversal of the sub-trees continues till the sub-trees are found to be Null. Here, commas have been used as delimiters for the sake of improved readability.

The post-order and in-order may be similarly defined using eqn. 2 and eqn. 3 respectively.

PostOrder(T) = PostOrder(T_L), PostOrder(T_R), R … eqn. 2

InOrder(T) = InOrder(T_L), R, InOrder(T_R) … eqn 3

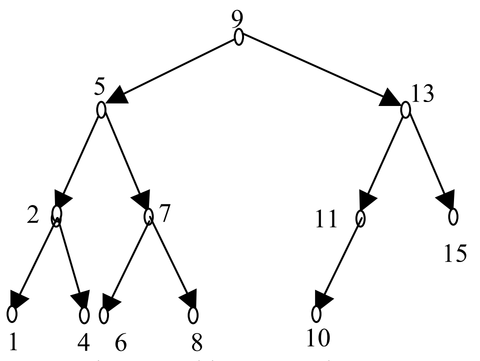

Fig. 17: A binary search tree

e.g., the tree on figure 17 is a binary search tree. The pre-order, post-order and in-order traversal outputs for the BST are given below in the respective order.

Pre-order output: 9, 5, 2, 1, 4, 7, 6, 8, 13, 11, 10, 15

Post-order output: 1, 4, 2, 6, 8, 7, 5, 10, 11, 15, 13, 9

In-order output: 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 13, 15

1.6 Discrete Probability [Chapter 7, Rosen-2006]

Probability is the mathematical description of randomness. Basic definition of probability and randomness has been defined in section 1.4 of this article. Let us start with the concepts behind probability distribution and discrete probability here.

A *probability model* is a mathematical description of a random phenomenon consisting of two parts: a sample space S and a way of assigning probabilities to events. The sample space defines the set of all possible outcomes whereas an event is a subset of a sample space representing a possible outcome or a set of outcomes.

A *random variable* is a function or rule that assigns a number to each outcome. Basically it is just a symbol that represents the outcome of an experiment.

e.g., let X be the number of heads when the experiment is flipping a coin n times. Similarly, S be the speed of a car registered on a radar detector on highway.

The values for a random Variable could be discrete or continuous depending on the experiment.

A *discrete random variable* can hold all possible outcomes without missing any, although it might take an infinite amount of time.

A *continuous random variable* is used to measure an uncountable number of values even if infinite amount of time is given.

e.g., if a random variable X represents an outcome which is a natural number between 1 and 100, then X may have infinite number of values. One can never list all possible outcomes for X even if infinite amount of time is allowed. Here, X is a continuous random variable. On the contrary, for the same interval of 1 to 100, another random variable Y can be used is to list all the integer values in the range. Here, Y is a discrete random variable.

An upper-case letter, say X, will represent the ***name*** of the random variable. Its lower-case counterpart, x, will represent the ***value*** of the random variable.

The probability that the random variable **X** will equal x is:

P(X = x) or more simply P(x).

A *probability distribution (density) function* is a table, formula, or graph that describes the values of a random variable and the probability associated with these values.

Probabilities associated with discrete random variables have the following properties:

i. $0 \le P(x) \le 1$ for all x

ii. $\Sigma P(x) = 1$

A discrete probability distribution can be represented an discrete random variable.

X	1	2	3	4	5	6
P(x)	1/6	1/6	1/6	1/6	1/6	1/6

Fig. 18: A discrete probability function for a rolling die

The *mean* μ of a probability distribution model is the sum of the product terms for individual events and its outcome probability. In other words, for the possible outcomes $x_1, x_2, \ldots, x_n$ in a sample space S if p_k is the probability of outcome x_k, then the mean of this probability would be $\mu = x_1 p_1 + x_2 p_2 + \ldots + x_n p_n$.

e.g., for the mean of the probability density for the distribution in figure 18 would be:

1*(1/6)+2*(1/6)+3*(1/6)+ 4*(1/6)+5*(1/6)+6*(1/6)

= 21*(1/6) = 3.5

The *variance* σ^2 of a discrete probability model is: $\sigma^2 = (x_1 - \mu)^2 p_1 + (x_2-\mu)^2 p_2 + \ldots + (x_k - \mu)^2 p_k$. The *standard deviation* σ is the square root of the variance.

e.g., for the probability distribution in figure 18, the variation σ^2 would be

$\sigma^2 = [(1- 3.5)^2*(1/6) + (2- 3.5)^2*(1/6) + (3- 3.5)^2*(1/6) + (4 - 3.5)^2*(1/6) + (5- 3.5)^2*(1/6) + (6 - 3.5)^2*(1/6)]$

= (6.25+2.25+0.25+0.5+2.25+6.25)*(1/6)

= 17.5*(1/6)

= 2.90

$\therefore$ standard deviation $\sigma = \sqrt{2.9} = 1.70$

These numbers indeed aims to derive the average value from repeated experiments. This is based on the single most important phenomenon of probability, i.e., the average value from repeated experiments is likely to be close to the expected value of one experiment. Moreover, it gets more likely to be closer as the number of experiments increases.

1.7 Finite State Machines [Chapter 13, Rosen-2011]

A computer system may be abstracted as a mapping from state to state driven by inputs. In other words, a system may be considered as a transition function T: $S \times I \rightarrow S \times O$, where S is the set of states and I, O are the input and output functions.

If the state set S is finite (not infinite), the system is called a *finite state machine* (FSM).

Alternately, a finite state machine (FSM) is a mathematical abstraction composed of a finite number of states, and transitions between those states. If the domain S×I is reasonably small, then one can specify T explicitly by diagrams similar to a flow graph to illustrate the way logic flows for different inputs. However, this is practical only for machines that have a very small information capacity.

An FSM has a finite internal memory, an input feature that reads symbols in a sequence and one at a time, and an output feature.

The operation of an FSM begins from a *start state*, goes through transitions depending on input to different states and can end in any valid state. However, only a few of all the states mark a successful flow of operation. These are called *accept states*.

The information capacity of an FSM is C = log |S|. Thus, if we represent a machine having an information capacity of C bits as an FSM, then its state transition graph will have $|S| = 2^C$ nodes.

A *finite-state machine* is formally defined as $M = (S, I, O, f, g, s_0)$

S is the state set;

I is the set of input symbols;

O is the set of output symbols;

f is the state transition function;

g is the output function;

and s_0 is the initial state.

Given an input $x \in I$, on state S_k, the FSM makes a transition to state S_h following state transition function f and produces an output $y \in O$ using the output function g

e.g., figure 19 illustrates a FSM with S_0 as the start state and S_1 as the final state. Here, S = {S_0, S_1, S_2}; I = {0, 1}; O = {2, 3}; f(S_0, 0) = S_2, f(S_0, 1) = S_1, f(S_1, 0) = S_2, f(S_1, 1) = S_2, f(S_2, 0) = S_2, f(S_2, 1) = S_0; g(S_0, 0) = 3, g(S_0, 1) = 2, g(S_1, 0) = 3, g(S_1, 1) = 2, g(S_2, 0) = 2, g(S_2, 1) = 3.

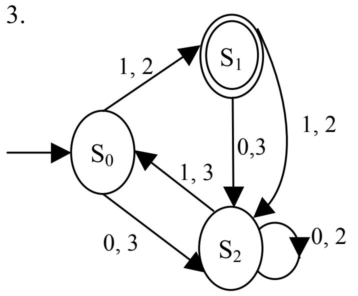

Fig. 19: Example of a FSM

The state transition and output values for different inputs on different states may be represented using a state table. The state table for the FSM in figure 19 is shown in figure 20. Each pair against an input symbol represents the new state, and the output symbol.

Current State	Input Symbols	
	0	1
S_0	S_2, 3	S_1, 2
S_1	S_2, 3	S_2, 2
S_2	S_2, 2	S_0, 3

(a)

Current State	Output f		State Trans g	
	Input Symbols		Input Symbols	
	0	1	0	1
S_0	3	2	S_2	S_1
S_1	3	2	S_2	S_2
S_2	2	3	S_2	S_0

(b)

Fig. 20: Tabular representation of FSM

e.g., Figure 20 (a) and 20(b) are two alternate representations of the FSM in figure 19.

1.8 Grammars [Chapter 13, Rosen-2011]

The grammar of a natural language tells us whether a combination of words makes a valid sentence. Unlike natural languages, a formal language is specified by a well-defined set of rules for syntaxes. The valid sentences of a formal language can be described by a grammar with the help of these rules, referred as *production rules*.

A *formal language* is a set of finite-length words or strings over some finite alphabet and a *grammar* specify the rules for formation of these words or strings. The entire set of words that are valid for a grammar, constitute the *language* for the grammar. Thus, the grammar G is any compact, precise mathematical definition of a language L as opposed to just a raw listing of all of the language's legal sentences, or just examples of them.

A grammar implies an algorithm that would generate all legal sentences of the language. There are different types of grammars.

A *phrase-structure* or *Type-0* grammar G = (V, T, S, P) is a 4-tuple, in which:

- V is the vocabulary i.e., set of words.
- $T \subseteq V$ is a set of words called terminals
- $S \in N$ is a special word called the start symbol.
- P is the set of productions rules for substituting one sentence fragment for another.

There exists another set $N = V - T$ of words called non-terminals. The non-terminals represent concepts like *noun.* Production rules are applied on strings containing non-terminals, until no more non-terminal symbols are present in the string. The start symbol S is a non-terminal.

The *language* generated by a formal grammar G, denoted by L(G), is the set of all strings over the set of alphabets V that can be generated, starting with the start symbol, by applying production rules until no more non-terminal symbols are present in the string.

e.g., let G = ({S, A, a, b}, {a, b}, S, {$S \to aA$, $S \to b$, $A \to aa$}). Here, the set of terminals are N={S, A}, where S is the start symbol. The three production rules for the grammar are given as P1: $S \to aA$; P2: $S \to b$; P3: $A \to aa$.

Applying the production rules in all possible way, the following words may be generated from the start symbol.

$S \to aA$	(using P1on start symbol)
$\to aaa$	(using P3)
$S \to b$	(using P2on start symbol)

Nothing else can be derived for G. Thus the language of the grammar G consists of only two words L(G) = {aaa, b}.

1.8.1 Language Recognition:

Formal grammars can be classified according to the types of productions that are allowed. The Chomsky hierarchy describes such classification scheme. This has been introduced by Noam Chomsky in 1956.

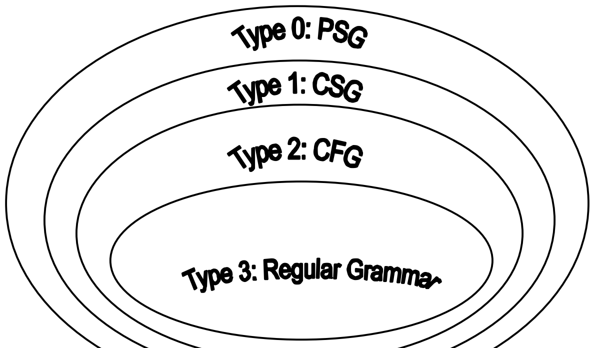

Fig. 21: Chomsky Hierarchy of Grammars

As illustrated in Figure 21, we infer the following on different types of grammars:

1. Every regular grammar is a context free grammar (CFG).
2. Every CFG is a context sensitive grammar (CSG).
3. Every CSG is a phrase structure grammar (PSG).

Context Sensitive Grammar: All fragments in the RHS are either longer than the corresponding fragments in the LHS, or empty, i.e., if $b \to a$, then $|b| < |a|$ or $a = \phi$.

A formal language is context-sensitive if there is a context-sensitive grammar generates it.

Context Free Grammar: All fragments in the LHS are of length 1, i.e., if $A \to a$, then $|A| = 1$ for all $A \in N$.

The term context-free derives from the fact that A can always be replaced by a, regardless of context in which it occurs.

A formal language is context-free if there is a context-free grammar generates it. Context-free languages are the theoretical basis for the syntax of most programming languages.

Regular Grammar: All fragments in the RHS are either single terminals, or its a pair built by a terminal and a non-terminal, i.e., if $A \to a$, then either $a \in T$, or a = cD, or a =Dc for $c \in T$, $D \in N$.

If a=cD, then the grammar is called a right linear grammar. On the other hand, if a=Dc, then the grammar is called a left linear grammar. Both the right linear or left linear grammars are regular or Type-3 grammar.

The language L(G) generated by a regular grammar G is called a regular language.

A regular expression A is a string (or pattern) formed from the following six pieces of information: $a \in \Sigma$, the set of alphabets, ε, 0 and the operations, OR (+), PRODUCT (.), CONCATENATION (*). The language of G, L(G) is equal to all those strings which match G, $L(G) = \{x \in \Sigma^* | \text{ x matches G}\}$.

For any $a \in \Sigma$, L(a) = a; L(ε) = {ε}; L(0) = 0.

+ functions as an or, $L(A + B) = L(A) \cup L(B)$.

. creates a product structure, L(AB) = L(A).L(B).

* denotes concatenation, $L(A^*) = \{x_1x_2 \dots x_n \mid x_i \in L(A) \text{ and } n \geq 0\}$

e.g., the regular expression (ab)* matches the set of strings: {ε, ab, abab, ababab, abababab, . . .}.

e.g., the regular expression (aa)* matches the set of strings on one letter *a* which have even length.

e.g., the regular expression (aaa)*+(aaaaa)* matches the set of strings of length equal to a multiple of 3 or 5.

1.9 Numerical precision [Chapter 2, Cheney-2007]

The main goal of numerical analysis is to develop efficient algorithms for computing precise numerical values of functions, solutions of algebraic and differential equations, optimization problems, etc.

A matter of fact is that all digital computers can only store finite numbers. In other words, there is no way that a computer can represent an infinitely large number, be it an integer, or a rational number, or any real or all complex numbers[definitions: in section

1.10]. So the mathematics of approximation becomes very critical to handle all the numbers in the finite range that computers can handle.

Each number in a computer is assigned a location or word, consisting of a specified number of binary digits or bits. A k bit word can store a total of $N = 2^k$ different numbers.

e.g., a computer that uses 32 bit arithmetic can store a total of $N = 2^{32} \approx 4.3\times10^9$ different numbers, while another one that uses 64 bits, can handle $N' = 2^{64} \approx 1.84\times10^{19}$ different numbers. The question is how to distribute these N numbers over the real line for maximum efficiency and accuracy in practical computations.

One evident choice is to distribute them evenly, leading to fixed point arithmetic. In this system, the first bit in a word is used to represent a sign, and the remaining bits are treated for integer values. This would allow to represent the integers from $1- ½N$, i.e., $= 1- 2^{k-1}$ to 1. As an approximating method, this is not good for the non-integer numbers.

Another option is to space the numbers closely together, say with uniform gap of 2^{-n}, and so distribute the total N numbers uniformly over the interval $-2^{-n-1}N < x \leq 2^{-n-1}N$. Real numbers lying between the gaps are represented by either *rounding*, meaning the closest exact representative, or by *chopping*, meaning the exact representative immediately below (or above if negative) the number.

Numbers lying beyond the range must be represented by the largest (or largest negative) representable number. This becomes a symbol for overflow. Overflow occurs when a computation produces a value larger than the maximum value in the range.

When processing speed is a significant bottleneck, the use of the fixed point representations is an attractive and faster alternative to the more cumbersome floating point arithmetic most commonly used in practice.

Let's define a couple of very important terms *accuracy* and *precision* associated with numerical analysis.

Accuracy is the closeness with which a measured or computed value agrees with the true value.

Precision, on the other hand, is the closeness with which two or more measured or computed values for the same physical substance agree with each other. In other words, precision is the closeness with which a number represents an exact value.

Let x be a real number and let x* be an approximation. The *absolute error* in the approximation $x^* \approx x$ is defined as $| x^* - x |$. The *relative error* is defined as the ratio of the absolute error to the size of x, i.e., $| x^* - x | / | x |$, which assumes $x \neq= 0$; otherwise relative error is not defined.

e.g., 1000000 is an approximation to 1000001 with an absolute error of 1 and a relative error of 10^{-6}, while 10 is an approximation of 11 with an absolute error of 1 and a relative error of 0.1. Typically, relative error is more intuitive and the preferred determiner of the size of the error. The present convention is that errors are always ≥ 0, and are $= 0$ if and only if the approximation is exact.

An approximation x* has k *significant decimal digits* if its relative error is $< 5 \times 10^{-k-1}$. This means that the first k digits of x* following its first nonzero digit are the same as those of x.

Significant digits are the digits of a number that are known to be correct. In a measurement, one uncertain digit is included.

e.g., measurement of length with a ruler of 15.5 mm with ±0.5 mm maximum allowable error has 2 significant digits, whereas a measurement of the same length using a caliper and recorded as 15.47mm with ±0.01 mm maximum allowable error has 3 significant digits.

1.10 Number Theory [Chapter 4, Rosen-2011]

Number theory is one of the oldest branches of pure mathematics, and one of the largest. Of course, it concerns questions about numbers, usually meaning whole numbers and fractional or rational numbers. The different types of numbers include integer, real number, natural number, complex number; rational numbers, etc.

1.10.1 Divisibility: Let's start this section with a brief description of each of the above types of numbers, starting with the Natural Numbers.

Natural Numbers: This group of numbers starts at 1 and it continues like 1, 2, 3, 4, 5, and so on. Zero is not in this group. It has no negative or fractional numbers in the group of natural numbers. The common mathematical symbol for the set of all natural numbers is **N.**

Whole Numbers: This group has all of the Natural Numbers in it plus the number 0.

Unfortunately, the definitions of natural and whole numbers as given above are not universally accepted by all. There seems to be no general agreement about whether to include 0 in the set of natural numbers. In fact, Ribenboim (1996) states: *Let P be a set of natural numbers; whenever convenient, it may be assumed that $0 \in P$!*

Many mathematicians consider that traditionally in Europe, the sequence of natural numbers started with 1 (0 was not even considered to be a number by the Greek). In the 19th century, set theoreticians and other mathematicians started the convention of including 0 in the set of natural numbers.

Integers: This group has all the Whole Numbers in it and their negatives. The common mathematical symbol for the set of all integers is **Z**, i.e., **Z** = {…, -3, -2, -1, 0, 1, 2, 3…}.

Rational Numbers: These are any numbers that can be expressed as a ratio of two integers. The common symbol for the set of all rational numbers is **Q.**

The rational numbers may be classified into three types based on how the decimals act. The decimals either do not exist, e.g., as in 15. When decimals exist, it may terminate, as in 15.6; or the decimals repeat with a pattern, as in 1.666..., (which is 5/3).

Irrational Numbers: These are numbers that can not be expressed as an integer divided by an integer. These numbers have decimals that never terminate and never repeat with a pattern: e.g., PI or $\sqrt{2}$

Real Numbers: This group is made up of all the Rational and Irrational Numbers. The numbers that are encountered when studying algebra are real numbers. The common mathematical symbol for the set of all real numbers is **R.**

Imaginary Numbers: These are all based on the imaginary number *i*. This imaginary number is equal to the square root of -1. Any real number multiple of *i* is an imaginary number; e.g., *i*, 5*i*, 3.2*i*, -2.6*i* etc.

Complex Numbers: A Complex Number is a combination of a real number and an imaginary number in the form a + b*i*. The real part is a, and b is called the imaginary part. The common mathematical symbol for the set of all complex numbers is **C.**

e.g., 2 + 3*i*, 3 -5*i*, 7.3 + 0*i*, and 0 + 5*i*.

Consider the last two examples:

7.3 + 0*i* is same as the real number 7.3. Thus all real numbers are complex numbers with zero for the imaginary part.

Similarly, 0 + 5*i* is just the imaginary number 5*i*. Thus, all imaginary numbers are complex numbers with zero for the real part.

Elementary number theory involves divisibility among integers. Let a, b∈Z with a≠0. The expression a|b i.e., *a divides b* if ∃c∈Z: b=ac i.e., there is an integer c such that c times a equals b.

e.g., 3|−12 is True, but 3|7 is False.

If a divides b, then we say that *a* is a factor of *b* or *a* is divisor of *b*, and *b* is a multiple of *a*.

b is even if and only if 2|b.

Let a, d ∈ Z with d > 1. Then *a mod d* denotes that the remainder r from the division algorithm with dividend *a* and divisor d; i.e. the remainder when a is divided by d. We can compute (a mod d) by: a − d*⌊a/d⌋, where ⌊a/d⌋ represents the floor of the real number.

Let Z^+ = {n∈Z | n>0} and a, b∈Z, m∈Z^+, then a is congruent to b modulo m, written as *a≡b (mod m)*, if and only if m | a−b.

Alternately, a is congruent to b modulo m iff (a−b) mod m = 0.

1.10.2 Prime number, GCD: An integer p>1 is prime iff it is not the product of any two integers greater than 1, i.e., p is prime if p>1 ∧ ∃ ¬ a, b∈N: a>1, b>1, a*b=p.

The only positive factors of a prime p are 1 and p itself. e.g., the numbers 2, 13, 29, 61, etc. are prime numbers. Non-prime integers greater than 1 are called composite numbers. A composite number may be composed by multiplying two integers greater than 1.

There are many interesting applications of prime numbers. This includes *public-key cryptography* scheme involving exchange of *public keys* containing the product *p*q* of two random large primes *p* and *q* (a *private key*) that must be kept secret by a given party.

The *greatest common divisor* gcd(a, b) of integers a, b is the greatest integer d that is a divisor both of a and of b, i.e.,

d = gcd(a, b) for max(d: d|a ∧ d|b)

e.g., gcd(24, 36) = 12.

Integers a and b are called relatively prime or co-prime if and only if their GCD is 1.

e.g., neither 35 and 6 are prime, but they are co-prime as these two numbers have no common factors greater than 1, so their GCD is 1.

A set of integers X={i_1, i_2,…} is relatively prime if all possible pairs i_h, i_k, h≠k, drawn from the set X are relatively prime.

1.11 Algebraic Structures

This section introduces a few representations used in higher algebra. An algebraic structure consists of one or two sets closed under some operations and satisfying a number of axioms, including none.

e.g., group, monoid, ring, lattice etc are examples of algebic structures. Each of these has been defined later in this section.

1.11.1 Group: A set S closed under a binary operation • forms a group if the binary operation satisfies the following 3 criteria:

Associative: ∀ a, b, c ∈ S, the equation (a • b) • c = a • (b • c) holds.

Identity: There exist an identity element I∈S such that for all a∈S, I • a = a • I = a.

Inverse: Every element a ∈ S, has an inverse a' ∈ S with respect to the binary operation, i.e.. a • a' = I.

e.g., the set of integers **Z** with respect to the addition operation is a group. The identity element of the set is 0 for the addition operation. ∀x ∈ Z, the inverse of x would be –x which is also included in Z.

Closure property: ∀ a, b ∈ S, the result of the operation a • b ∈ S.

A group that is commutative i.e., a • b = b • a, is known as a commutative or Abelian monoid [defined later this section]. However, the set of natural numbers **N** (with the operation of addition) is not a group, since there is no inverse for any x > 0 in the set of natural numbers. Thus the third rule of inverse for our operation is violated. However, the set of natural number has some structure.

Sets with an associative operation (the first condition above) are called semigroups, and if they also have an identity element (the second condition) then they are called monoids.

Our set of natural numbers under addition is then an example of a monoid, a structure that is not quite a group because it is missing the requirement that every element have an inverse under the operation.

A monoid is a set S that is closed under a single associative binary operation • and has an identity element I∈S such that for all a∈S, I • a = a • I = a. A monoid must contain at least one element.

e.g., the set of natural numbers **N** form a commutative monoid under addition with identity element 0. The same set of natural numbers **N** also forms a monoid under multiplication with identity element 1. The set of positive integers **P** form a commutative monoid under multiplication with identity element 1.

It may be noted that unlike a group, elements of a monoid need not have inverses. It can also be thought of as a semi-group with an identity element.

A *subgroup* is a group *H* contained within a bigger one, *G such that* the identity element of *G* is contained in *H*, and whenever h_1 and h_2 are in *H*, then so are $h_1 \bullet h_2$ and $h_1^{-1.}$ Thus, the elements of *H*, equipped with the group operation on *G* restricted to *H*, form indeed a group.

Given any subset *S* of a group *G*, the subgroup generated by *S* consists of products of elements of *S* and their inverses. It is the smallest subgroup of *G* containing *S*.

e.g., let *G* be the Abelian group whose elements are *G* ={0, 2, 4, 6, 1, 3, 5, 7} and whose group operation is addition modulo 8. This group has a pair of nontrivial subgroups: *J*={0, 4} and *H*={0, 2, 4, 6}, where *J* is also a subgroup of *H*.

In group theory, a cyclic group is a group that can be generated by a single element, in the sense that the group has an element *a* (called *generator* of the group) such that, when written multiplicatively, every element of the group is a power of *a*.

A group G is cyclic if G = {a^n for any integer n}.

Since any group generated by an element in a group is a subgroup of that group, showing that the only subgroup of a group G that contains *a* is G itself suffices to show that G is cyclic.

e.g., the group *G* = {0, 2, 4, 6, 1, 3, 5, 7} with respect to addition modulo 8 operation is cyclic. The subgroups *J*={0, 4} and *H*={0, 2, 4, 6}, are also cyclic.

1.11.2 Rings: If we take an Abelian group and define a second operation on it, a new structure is found that is different from just a group. If this second operation is associative, and it is distributive over the first then we have a ring.

A ring is a triple of the form (S, +, •), where (S, +) is an Abelian group (S, •) is a semi-group and • is distributive over +; i.e., ∀ a, b, c ∈ S, the equation $a \bullet (b + c) = (a \bullet b) + (a \bullet c)$ holds. Further, if • is commutative, then the ring is said to be commutative. If there is an identity element for the • operation, then the ring is said to have an identity.

e.g., (**Z**, +, *), i.e., the set of integers Z, with the usual addition and multiplication operations is a ring. As (Z, *) is commutative, this ring is a commutative or Abelian ring. The ring has 1 as its identity element.

Let's note that the second operation may not have an identity element, nor do we need to find an inverse for every element with respect to this second operation. As for what distributive means, intuitively it is what we do in elementary mathematics when perform the following change: a * (b + c) = (a * b) + (a * c).

A field is a ring for which the elements of the set, excluding 0, form an Abelian group with the second operation.

e.g., a simple example of a field is the field of rational numbers (**R**, +, *) with the usual addition and multiplication operations. The numbers of the format $a/b \in \mathbf{R}$, where *a, b* are integers, and $b \neq 0$. The additive inverse of such a fraction is simply $-a/b$, and the multiplicative inverse is *b/a* provided that $a \neq 0$.

Acknowledgment

The author thankfully acknowledges the contribution of Prof. Arun Kumar Chatterjee, Ex-Head, Department of Mathematics, Manipur University, India and Prof. Devadatta Sinha, Ex-Head, Department of Computer Science & Engineering, University of Calcutta, India in preparing this chapter on Mathematical Foundations.

References

[1] E. Ward Cheney and David R. Kincaid, Numerical Mathematics and Computing (Hardcover), 6th Edition, Addison Wesley, 784 pages, ISBN: 978-0495114758.

[2] Kenneth Rosen, Discrete Mathematics and its Applications, 7th Edition, McGraw-Hill, 2011, 1072 pages, ISBN-13: 978-0073383095

Matrix of Topics vs. Reference material

	Cheney - 2007	Rosen - 2011
Sets, Relations, Functions		*
Basic Logic		*
Proof techniques		*
Basic counting		*
Graphs and Trees		*
Discrete probability		*
Finite State Machines		*
Grammars		*
Numerical precision	*	
Number theory		*
Algebric structures		

Recommended References for Mathematical Foundations

[Cheney-2007] E. Ward Cheney and David R. Kincaid, Numerical Mathematics and Computing (Hardcover), 6th Edition, Addison Wesley, 784 pages, ISBN: 978-0495114758.

[Rosen-2011] Kenneth Rosen, Discrete Mathematics and its Applications, 7th Edition, McGraw-Hill, 2011, 1072 pages, ISBN-13: 978-0073383095.

Appendix A. List of Further Readings

[Landin-1989] Joseph Landin, "An Introduction to Algebraic Structures"; Dover Publications, 1989, ISBN: 0486659402

[Pinter-2010] Charles C Pinter, A Book of Abstract Algebra, 2 ed., 2010, ISBN-10: 0486474178, ISBN-13: 978-0486474175.

[Cameron-2008] Peter J. Cameron, Introduction to Algebra, Oxford. University Press, 2 ed., 2008, ISBN-10: 0198569130, ISBN-13: 978-0198569138.

[Lipschutz-2009] Seymour Lipschutz, Marc Lipson Schaum's Outline of Discrete Mathematics; McGraw-Hill; 3 ed., 2009, ISBN-10: 0071615865, ISBN-13: 978-0071615860

[Bender-2005] Edward A. Bender, S. Gill Williamson, A Short Course in Discrete Mathematics, Dover Publications, 2005, ISBN 0-486-43946-1.

[Chartrand-1984] Gary Chartrand, Introductory Graph Theory, Dover Publications; ISBN-10: 0486247759, ISBN-13: 978-0486247755.

[Deo-974] Narsingh Deo, Graph Theory with Applications to Engineering and Computer Science; Prentice-Hall Inc. Upper Saddle River, NJ, USA,1974. ISBN: 0133634736

[Kolman-2008] Bernard Kolman, Robert Busby, Sharon Ross, Discrete Mathematical Structures, Pearson Educations, International Edition, 6th Ed., 2008, ISBN13: 9780132078450, ISBN10: 0132078457