AKSHAT SOOD

FOUNDATIONS OF COMPUTING

PROPOSITIONAL LOGIC

→ An argument is logically correct if in every situation that makes all premises true, the conclusion is true as well. (∀)

→ To prove that an argument is logically incorrect, show 1 situation where premises are true but the conclusion is false (∃)

RULES OF INFERENCE :

$\frac{p \vee q \quad \neg p}{q}$ $\frac{p \quad q}{p \wedge q}$ $\frac{p \wedge q}{p}$ $\frac{p}{p \vee q}$

$\frac{p \to q \quad p}{q}$ Modus Ponens $\frac{p \to q \quad \neg q}{\neg p}$ Modus Tollens $\frac{p \to q \quad q \to r}{p \to r}$ $\frac{p \to r \quad q \to r}{(p \vee q) \to r}$ *

$P \equiv Q$ iff both $\frac{P}{Q}$ and $\frac{Q}{P}$ are correct

PREDICATE LOGIC

→ Statements involving variables

→ By assigning a value to its variable, $P(x)$ becomes a proposition that has a truth value.

- Universal Quantification
  - → $\forall x P(x)$, denotes $P(x)$ is true for all values of x from the domain
  - * to prove that $\forall x P(x)$ is false, find a single value c, for which $P(x)$ is false.
  - Universal Instantiation : $\frac{\forall x P(x)}{P(c)}$
  - Universal Generalisation : where c is an arbitrary entity from the domain. $\frac{P(c)}{\forall x P(x)}$
- Existential Quantification : $\exists x P(x)$
  - → there exists a value for x in the domain such that $P(x)$ is true
  - * To prove that $\exists x P(x)$ is true, prove that $P(c)$ is true
  - Existential Generalisation : $\frac{P(c)}{\exists x P(x)}$

PROOF BY CONTRADICTION:- We can prove that p is true if we can show that $\neg p \to (q \wedge \neg q)$ is true for some q

DE MORGANS LAWS → $\frac{\neg \forall x P(x)}{\exists x \neg P(x)}$ $\frac{\forall x \neg P(x)}{\neg \exists x P(x)}$ $\frac{\exists x \neg P(x)}{\neg \forall x P(x)}$ $\frac{\neg \exists x P(x)}{\forall x \neg P(x)}$

SETS

→ Collection of things called its elements.

$x \in S$: x is an element of S ; $x \notin S$: x is not an element of S

* Anything can be an element of a set
* A set can be an element of a set
* Repeated occurences of elements don't matter, Order of listing doesn't matter.
* Empty Set : $\emptyset \equiv \{\}$. Set with no elements. Can be a part of another set.
* 2 sets are equal if they have the same elements (order and listing doesn't matter)

- Z (Integers) $= \{\ldots -3, -2, -1, 0, 1, 2, 3 \ldots\}$
- N (Natural Numbers) $= \{0, 1, 2 \ldots\}$
- Q : Rational Numbers, R : Real Numbers, N^+ : Positive Natural, R^+ : Positive Real

⇒ SUBSET : $B \subseteq A$ → B is a subset of A if every element in B is also an element of A

⇒ PROPER SUBSET : $B \subset A$ → B is a proper subset of A if there is an element in A which is not in B → $B \subset A$ iff $B \subseteq A$ and $B \neq A$

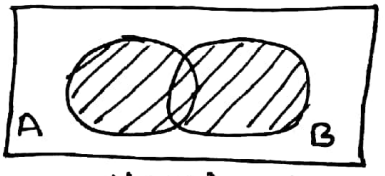
* To prove $A = B$, prove that $A \subseteq B$ and $B \subseteq A$.
* To prove $A \neq B$, find an element in A which is not in B or an element in B which is not in A.

⇒ **POWER SET** $P(S)$: The set of all subsets of a set S.

* $P(S) = \{A \mid A \subseteq S\}$ * $\emptyset$ and $S \in P(S)$

⇒ **UNION**: $A \cup B$ consists of those elements which are either in A or B or both

* $A \cup B = \{x \mid x \in A \text{ (inclusive) or } x \in B\}$

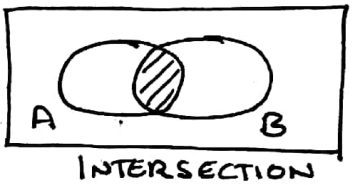


UNION

* IDENTITY LAW: $A \cup \emptyset = A$
* COMMUTATIVE LAW: $A \cup B = B \cup A$
* ASSOCIATIVE LAW: $(A \cup B) \cup C = A \cup (B \cup C)$
* IDEMPOTENT LAW: $A \cup A = A$ * $A \subseteq B$ iff $A \cup B = B$

⇒ **INTERSECTION**: $A \cap B$ consists of those elements which are both in A and B only

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

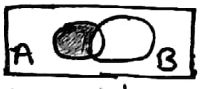


INTERSECTION

* DOMINATION LAW: $A \cap \emptyset = \emptyset$
* COMMUTATIVE LAW: $A \cap B = B \cap A$
* ASSOCIATIVE LAW: $(A \cap B) \cap C = A \cap (B \cap C)$
* IDEMPOTENT LAW: $A \cap A = A$ * $A \subseteq B$ iff $A \cap B = A$
* DISTRIBUTIVE LAWS: $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

⇒ **DIFFERENCE** (−) : A−B consists of those elements that are in A but not in B

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



DIFFERENCE

⇒ **COMPLEMENT** : $\bar{A}$ consists of those elements which are not in A

* $\bar{A} = \{x \in U \mid x \notin A\} = U - A$

COMPLEMENT

⇒ **PROPERTIES** : $A \cup \bar{A} = U$ $A \cap \bar{A} = \emptyset$ $\bar{\bar{A}} = A$ $\bar{U} = \emptyset$ $\bar{\emptyset} = U$

$\overline{A \cup B} = \bar{A} \cap \bar{B}$ $\overline{A \cap B} = \bar{A} \cup \bar{B}$ $A - B = A \cap \bar{B}$ $A \subseteq B$ iff $\bar{B} \subseteq \bar{A}$

SEQUENCES → List of things taken in a certain order.

* Order of listing does matter, repeated occurences do matter.
* k-tuple: sequence with k elements. * Ordered Pair: 2-tuple

CARTESIAN PRODUCTS OF SETS : $A \times B = \{(x, y) \mid x \in A \text{ and } y \in B\}$

* When you have n-sets with $E_1, E_2 \ldots\ldots E_n$ elements each, then the total number of n-tuples in their cartesian product will be $E_1 \times E_2 \times \ldots\ldots \times E_n$

BINARY RELATIONS : aRb or $R(a,b)$ → a is related to b if $(a,b) \in R$ and $a \not R b$ when $(a,b) \notin R$.

* They can be represented using directed graphs or by a 0-1 matrix.

PROPERTIES OF RELATIONS :-

① **REFLEXIVITY**: If $(a,a) \in R$ for every element $a \in A$

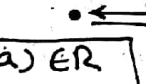
$\forall a \in S, (a,a) \in R$

② → A relation is irreflexive if $\forall a \in S, (a,a) \in R$

② **SYMMETRY** : $\forall a, b \in S, (a,b) \in R$ and $(b,a) \in R$.

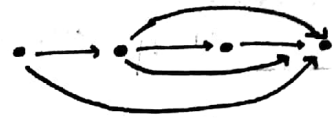
→ It is antisymmetric if $\forall a, b \in S, (a,b) \in R$ and $(b,a) \in R$ iff $a = b$

* Symmetric and antisymmetric are not opposites.


(no loops)


(Arrows clash)

③ TRANSITIVITY : $(a,b) \in R$ and $(b,c) \in R$, then $(a,c) \in R$ as well



* R is transitive if in the directional graph every 2 step journey can be done in a single step.
* If there are no 2 step journies → transitive by default.

TYPES OF RELATIONS

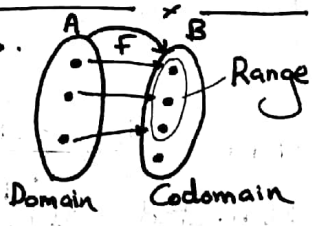
① EQUIVALENCE RELATIONS ⟷ Reflexive, symmetric, transitive

② PARTIAL ORDER → Reflexive, antisymmetric, transitive.

③ LINEAR ORDER → Partial order, $\forall a,b \in A$ either $(a,b) \in R$ or $(b,a) \in R$
* Name is so because the Hasse Diagram of a linear order is a line

HASSE DIAGRAM → Omit the loops, remove all arrows other than 1 steps, rearrange the dots so the arrows are only pointing upwards, replace arrow with lines.

FUNCTIONS → rule which associates elements of 2 sets.
* Every element of the domain has to be mapped
* One element cannot be mapped to 2 different places



$f: a \to b = \{(a \times b) \in A \times B \mid f(a) = b\}$

PROPERTIES OF FUNCTIONS

① ONE-TO-ONE (INJECTIVE) → Each element of the domain is mapped to a distinct element of the codomain. $\forall x, y \in A$, if $x \neq y$ then $f(x) \neq f(y)$

② ONTO (SURJECTIVE) → If the range is the same as the codomain.

③ BIJECTION → When the function is both one-to-one and onto.

COUNTING * Size of a set (Cardinality) = $|S|$, * $|\emptyset| = 0$

SUM RULE → If A and B are disjoint sets → $|A \cup B| = |A| + |B|$

INCLUSION EXCLUSION PRINCIPLE → $|A \cup B| = |A| + |B| - |A \cap B|$
↳ for non disjoint sets, ie sets which have elements in common.

PRODUCT RULE → k tasks, n_i ways to do the i^{th} task, then total number of ways to do all tasks → $n_1 \times n_2 \times n_3 \cdots \times n_k$

PIGEONHOLE PRINCIPLE → If $k \in N^+$ and k+1 or more objects are placed in k boxes, then there is atleast 1 box with 2 or more objects.
→ If n objects are placed into k boxes then there will be atleast 1 box containing at least $\lceil n/k \rceil$ objects ($\lceil \rceil$ depicts ceiling function).

COUNTING SELECTIONS : Ways of selecting k items from a sets of n items

	ORDER MATTERS (PERMUTATIONS)	ORDER DOESN'T MATTER (COMBINATIONS)
REPETITIONS NOT ALLOWED	$n \cdot (n-1) \cdots (n-k+1)$	$\binom{n}{k}$ or ${}^nC_k$
REPETITIONS ALLOWED	n^k	$\binom{k+n-1}{n-1}$ or $\binom{k+n-1}{k}$

PROPERTIES OF CHOOSE

$\binom{n}{k} = \frac{n \cdot (n-1) \cdots n(k+1)}{1 \cdot 2 \cdots k}$

$\binom{n}{k} = \binom{n}{n-k}$

⇒ PASCAL'S IDENTITY : $\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$ * Counting of rows and columns entries start from 0

PROBABILITY THEORY

* $P(\bar{E}) = 1 - P(E)$ [$E \rightarrow$ Event]

* $P(E_1 \cup E_2) = P(E_1) + P(E_2) - P(E_1 \cap E_2)$

* **CONDITIONAL PROBABILITY:-**

$$P(E|F) = \frac{P(E \cap F)}{P(F)}$$

Probability of event E occuring given that F has already occurred.

* **BASE RATE FALLACY** → Assuming that $P(E|H)$ and $P(H|E)$ are the same thing.

⇒ **INDEPEDENCE** → $P(E|F) = P(E)$ and $P(F|E) = P(F)$ if F and E are 2 independent events

↳ Condition → $P(E \cap F) = P(E) \cup P(F)$

⇒ **BAYE'S THEOREM**

$$P(F|E) = \frac{P(E|F)P(F)}{P(E|F)P(F) + P(E|\bar{F})P(\bar{F})}$$

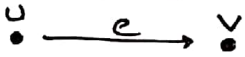
BAYESIAN SPAM FILTER

$$P(S|E_1 \cap E_2) = \frac{P(E_1|S)\,P(E_2|S)}{P(E_1|S)\,P(E_2|S) + P(E_1|\bar{S})P(E_2|\bar{S})}$$

GRAPHS

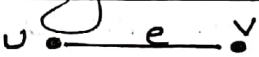
Simple Graph
Multigraph
Directed Graph

⇒ **Directed Graphs : Terminology**



→ u is adjacent to v
→ u is the initial or start vertex of e
→ v is the terminal or end vertex of e
→ **In degree** of v is the number of edges with v as their terminal vertex
→ **Out degree** of u is the number of edges with u as their initial vertex.
* A loop contributes to both.
* Number of Edges = Number of In-Degree Vertices = Number of out degree vertices

⇒ **Undirected Graph: Terminology**



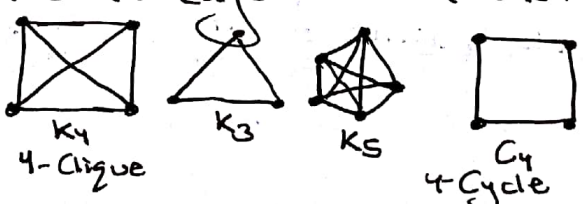
→ u and v are adjacent
→ e is incident with u and v
→ **Degree** of a vertex is the number of vertices incident with it.
→ If degree = 0 → **isolated vertex**
→ Degree = 1 → **Pendant vertex**
* Handshaking Theorem → $\text{Number of Edges} = \frac{\text{Sum of Degrees of Vertices}}{2}$

⇒ **PATHS** → Sequence of vertices
↳ **Length** → Number of edges in the path
→ **Simple Path** → Does not contain same edge twice.
→ **Hamiltanian Path** → Simple path passing through each vertex exactly once

⇒ **CYCLES** → Paths, beginning and ending with same vertex.
→ **Length**: Number of edges in it (Not counting first one twice)
→ **Simple Cycle** → Does not contain same edge twice.
→ **Hamiltonian Cycle** → Simple cycle passing through every vertex exactly once.

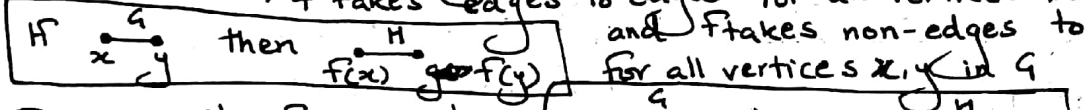
⇒ **COMPLETE GRAPH**: Simple graph that contains an edge between each pair of distinct vertices.

⇒ **CONNECTED GRAPH** → There is a path between every distinct vertex.



⇒ **ISOMORPHISM** → G and H are isomorphic if a function f between their vertice is such:→ f is a bijection (one-to-one and onto) [$f: G \to H$]

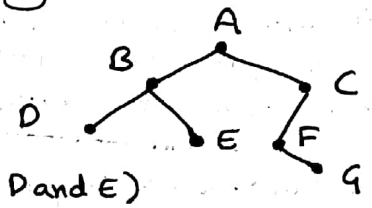
→ f takes edges to edges for all vertices in G and f takes non-edges to non edges for all vertices x, y in G



⇒ **INVARIENT** → property of a graph which is preserved in an isomorphic graph.

TREE → Connected simple graph with no simple cycles, such that adding an edge creates a cycle, and removing an edge make it unconnected.

⇒ **Rooted Tree** → Tree with a vertex as root.



⇒ **Terminology**:-

→ **Parent** → B is the parent of D

→ **Siblings** : Verteces with the same parent (Eg: D and E)

→ **Leaf** : Childless vertex (D, E, G)

→ **Internal** : Vertices with atleast one child (B, C, F, A)

→ **Level** : Length of the unique path from root to vertex. (Level(A) = 0, Level(D) = 2)

→ **Height** : Maximum level.

* A tree of height n is **balanced** if all its leaves are of level n or n-1.

SPECIAL TREES

⇒ **m-ary tree** : Every internal vertex has no more than m children.

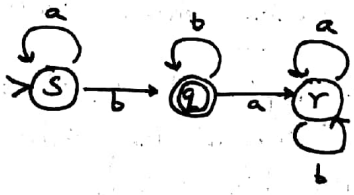
⇒ **full-m-ary tree** : If every internal vertex has exactly m-children.

* A full-m-ary tree of n internal vertices has (m·n)+1 total vertices.

⇒ **Linearly Ordered List** : A sequence of whose elements are linearly ordered.

* Lexicographical Order on words : $a \prec b \prec c \prec \dots \prec x \prec y \prec z$

FINITE AUTOMATA → finite-state machine



⇒ **Alphabet** : finite set s of symbols

⇒ **Word or String** : finite sequence of symbols from S

⇒ **Length of word** : Number of symbols in it.

⇒ **Concatenation** : Joining words

⇒ If w = xy, then x is **prefix** and y is **suffix** of w.

⇒ **Language** : Set of words over S.

⇒ > marks **initial state**, ◎ marks favourable state, arrows represent transitions.

⇒ **DFAS** → Automata in which there is a unique arrow coming out of the symbol.

REGULAR EXPRESSIONS → finite patter which represents an infinite language.

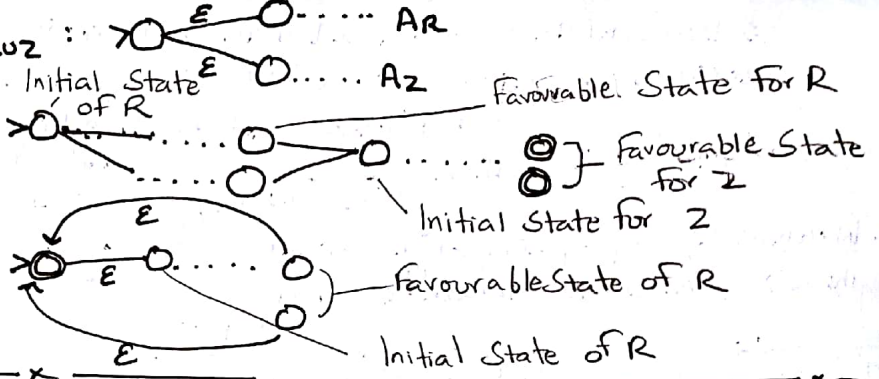
⇒ It is a string consisting of symbols from S and ∪, *, (,), ε, ∅.

⇒ A **regular language** is one which can be represented by a regular expression

* Not every language is Regular. ∴ Regular languages are those which are accepted by finite automata.

- If $R = \emptyset$, Language_of(R) = ∅, A_R: >○
- If $R = \varepsilon$, Language_of(R) = ε A_R: >◎
- If $R = a$, Language_of(R) = {a}, A_R: >○ —a→ ◎ [where A is a NFA]

- Language_of(RUZ) → A_{RUZ} :
- Language_of(RZ) A_{RZ} :
- Language_of(R*) A_{R^*} :



DFAs VS LANGUAGE

⇒ In an NFA there can be more than 1 computation on a word.
⇒ A word is accepted by an NFA if there is a computation that ends in a favourable state
⇒ NFA's reject words if → every computation is stuck
→ or ends up in a non-favourable state.
⇒ NFA does not increase computational power of finite automata.
⇒ SUBSET CONSTRUCTION : Converting NFAs to equivalent DFAs
* In DFA's → no choice, no 'getting stuck', no ε-jumps are allowed.

Things to Define:-
→ New States : Named after the subsets of the original NFA's states.
→ New Initial State: Set containing the original NFA's ~~subset~~ initial state and all states that are reachable from it by ε-jumps.
→ New Favourable States: Those subsets that contain at least one of the original NFA's favourable states.

→ Start from the initial state (new)
→ Make 2 arrows a and b from the state and map them to new states.
→ Repeat for all states until no new arrows can be formed.

* Worst case for increase in number of states when converting to DFA = $2^{\text{number of States in original NFA}}$

Other topics to Revise:-
→ Warshall's Algorithm
→ Tree Transversal
→ Monty Hall 3 door puzzle
→ Bayesian Spam Filters
→ Composition functions