CO-1

1. What are the AI problems? Explain with suitable examples.

Answer:
AI problems are those tasks which normally require human intelligence but can be solved by a computer system using algorithms and knowledge representation. These problems involve reasoning, learning, decision-making, and problem-solving.

Characteristics of AI problems:

1. **Complexity** – They often involve very large search spaces (e.g., chess with 10^120 possible states).
2. **Uncertainty** – AI problems deal with incomplete or noisy information.
3. **Goal-Oriented** – They require moving from an initial state to a defined goal state.
4. **Knowledge-Based** – Solving them needs domain-specific knowledge.
5. **Multiple Solutions –** AI problems may have more than one acceptable solution.

Examples:

- **Pathfinding** in Google Maps (finding shortest or fastest route).
- **Game Playing** such as chess and tic-tac-toe.
- **Medical Diagnosis** using symptoms and patient history.
- **Speech and Language Processing** like chatbots or translators.

Thus, AI problems are real-world tasks where machines attempt to simulate human-like intelligence for problem solving.

2. What are the underlying assumptions in AI problem solving?

Answer:
The **underlying assumption of AI** is that every intelligent task performed by humans can be **formulated as a computational process** and solved using algorithms and data structures.

Key assumptions include:

1. **Problem Representation** – Any problem can be represented using states, operators, and goals.
2. **Systematic Search** – By exploring the state space systematically, a solution can be found.
3. **Knowledge Usage** – AI assumes that domain knowledge, rules, and heuristics can guide the search efficiently.
4. **Rationality** – AI systems act rationally, i.e., they try to maximize performance measures.
5. **Balance of Optimality and Efficiency** – Solutions should be correct and practically computable.

Example:
In a chess-playing AI, it is assumed that the game can be represented as a tree of possible moves and outcomes, and algorithms like minimax with alpha-beta pruning can find the best move.

3. Explain AI techniques with example.

Answer:
AI techniques are methods used to represent, search, and reason about knowledge to solve problems.

Major techniques include:

1. **Search Techniques** – Exploring possible solutions systematically.
   - Example: A* search for pathfinding in maps.
2. **Knowledge Representation** – Representing facts and relationships in symbolic form.
   - Example: Predicate logic for expert systems.
3. **Reasoning Techniques** – Deductive, inductive, and probabilistic reasoning.
   - Example: Bayesian networks for medical diagnosis.
4. **Machine Learning** – Learning patterns from historical data.
   - Example: Neural networks for face recognition.

5. **Heuristic Methods** – Using approximate rules to guide problem solving.
    - Example: Chess evaluation function.
6. **Problem Decomposition** – Dividing a large problem into smaller sub-problems.

Thus, AI techniques combine **search, learning, and reasoning** to make problem solving efficient and intelligent.

4. What are the different level of AI models?

Answer:
AI systems can be studied at different levels, called **levels of models**, which define how knowledge and problem solving are structured.

1. **Knowledge Level** – Defines *what* the system knows about the problem and goals.
2. **Symbol Level** – Represents this knowledge using symbols, rules, or logic.
3. **Algorithm Level** – Provides step-by-step methods to process the symbols.
4. **Hardware Level** – Refers to the physical machine where AI runs (CPU, GPU).

Example: In chess:

- Knowledge level: Knowing the rules of the game.
- Symbol level: Representing the chess board as matrices.
- Algorithm level: Using minimax search with heuristics.
- Hardware level: Running on a computer.

This layered view helps in organizing AI problem solving systematically.

5. What are the criteria for success in AI problem solving?

Answer:
For an AI solution to be successful, it must meet several criteria:

1. **Correctness** – The solution should give accurate results.

2. **Efficiency** – It must use reasonable computation time and memory.
3. **Generality** – It should work for a wide variety of problems, not just one.
4. **Robustness** – Should work even when data is uncertain or incomplete.
5. **Optimality** – Should provide the best solution possible (e.g., shortest path).
6. **Scalability** – Must perform well even when the problem size increases.
7. **User Acceptance** – Should be practical and usable in the real world.

Example: A GPS navigation AI is successful if it consistently finds **fast, accurate, and robust routes** under changing traffic conditions.

6. Define the problem as a state space search with example.

Answer:
State space search is a fundamental way of representing AI problems, where we define a problem as a search through different states until the goal state is reached.

Components of state space search:

1. **Initial State** – The starting configuration of the problem.
2. **Operators/Actions** – Legal moves that change one state to another.
3. **State Space** – Set of all possible states reachable from the initial state.
4. **Goal State** – The target or solution state.
5. **Path Cost Function** – A measure of efficiency (distance, time, etc.).

Example (8-Puzzle Problem):

- Initial State: Tiles placed randomly.
- Operators: Move the blank tile up, down, left, or right.
- Goal State: Tiles arranged in numerical order.

Thus, many AI problems (like games, planning, robotics) are solved by state space search.

7. What is a Production System in AI? Explain its types.

Answer:
A **production system** is a model of computation in AI based on rules. It consists of:

1. **A set of production rules** – Condition-action pairs (IF-THEN rules).
2. **A working memory** – Stores facts about the current state.
3. **A control system** – Chooses which rule to apply next.

Types of Production Systems:

1. **Monotonic** – Rules never delete conditions (facts only increase).
2. **Non-Monotonic** – Rules can both add and remove facts.
3. **Deterministic** – Only one rule applies at each step.
4. **Non-Deterministic** – Multiple rules may apply.

Example (Medical Diagnosis):

- Rule: IF fever $\wedge$ cough THEN flu.
- Working memory: {fever, cough}.
- Control system infers: flu.

Production systems are widely used in **expert systems** and **knowledge-based systems**.

8. Explain Hill Climbing Search with its drawbacks.

Answer:
Hill climbing is a local search algorithm that continuously moves to the neighbor with the best evaluation function value, similar to climbing uphill.

Steps:

1. Start from an initial state.
2. Evaluate all neighbors.

3. Move to the neighbor with the highest value.
4. Repeat until no better neighbor exists.

Drawbacks:

1. **Local Maxima** – May get stuck in a peak that is not the global best.
2. **Plateau** – A flat area with no improvement; search gets stuck.
3. **Ridges** – Requires diagonal moves that simple algorithms may miss.

Example: Optimizing a mathematical function to find its maximum value.

Although simple and memory-efficient, hill climbing often fails for complex problems.

9. Explain Best-First Search with an example.

Answer:
Best-First Search is a heuristic search that explores the node which appears most promising according to a heuristic function h(n)h(n)h(n).

Algorithm steps:

1. Start with the initial node.
2. Place nodes in a priority queue based on h(n)h(n)h(n).
3. Expand the node with the lowest heuristic cost.
4. Continue until the goal is found.

Example: In a route-finding problem, best-first search uses **straight-line distance** to goal as the heuristic.

Advantage: Faster than blind search methods like BFS and DFS.
Disadvantage: May still fall into local traps or loops if not implemented carefully.

10. Explain A search algorithm.Why is it optimal ?

Answer:
The **A*** algorithm is an informed search strategy that combines the cost so far and the estimated cost to the goal:

f(n)=g(n)+h(n)

- g(n): Cost from start to node nnn.
- h(n): Estimated cost from nnn to goal.
- f(n): Total estimated cost of the path through nnn.

Optimality:
A* is **complete** (finds a solution if one exists) and **optimal** (finds the least-cost path) if the heuristic h(n) is **admissible** (never overestimates the true cost).

Example: In maps,

- g(n) = distance already traveled.
- h(n) = straight-line distance to destination.

A* is widely used in **pathfinding, robotics, and games** because of its efficiency and guaranteed optimality.

11. What is Problem Reduction in AI?

Answer:
Problem reduction means breaking down a large, complex problem into smaller, easier-to-solve sub-problems. Solutions to the sub-problems are then combined to solve the original problem.

Advantages:

1. Reduces computational complexity.
2. Makes problems more structured and manageable.
3. Encourages reuse of sub-problem solutions.

Examples:

- **Tower of Hanoi**: Solved by reducing nnn-disk problem into two n−1 disk problems.
- **Planning**: Cooking a meal broken into subtasks – chopping, boiling, serving.

Problem reduction is widely used in **divide and conquer algorithms** and **planning systems**.

12. Explain Simulated Annealing in AI?

Answer:
Simulated Annealing (SA) is a probabilistic optimization technique inspired by the annealing process in metallurgy, where metals are cooled slowly to reach a low-energy stable state.

Steps:

1. Start with an initial solution.
2. Pick a random neighbor solution.
3. If it is better, accept it.
4. If worse, accept it with probability:
   $P=e^{(-\Delta E/T)}$
   where ΔE = increase in cost, T = temperature.
5. Gradually reduce temperature according to a cooling schedule.

Advantages:

- Can escape local maxima by accepting worse solutions early on.
- Suitable for large, complex search spaces.

Applications:

- Travelling Salesman Problem (TSP).
- Scheduling and optimization problems.

Simulated Annealing is powerful because it balances **exploration (random moves)** and **exploitation (choosing better solutions)**.

CO-2

1. What is knowledge representation in AI ? Why is it important ?

Answer:
Knowledge Representation (KR) is the process of storing information about the world in a form

that a computer system can use to solve complex problems such as reasoning, learning, and decision making.

Importance of Knowledge Representation:

1. **Bridge between real world and AI systems** – It converts human knowledge into machine-usable format.
2. **Supports reasoning** – Allows AI systems to infer new facts from existing knowledge.
3. **Efficiency** – A good KR reduces complexity in problem solving.
4. **Communication** – Enables interaction between humans and AI in natural ways.
5. **Reusability** – Stored knowledge can be used across multiple tasks.

Example: In an expert medical system, knowledge about symptoms, diseases, and treatments is represented in rules:

- Rule: IF fever $\wedge$ cough THEN flu.

Thus, KR is fundamental to AI because it defines *what knowledge is stored, how it is represented, and how it is used in reasoning*.

2. What are the representations and mappings in knowledge representation?

Answer:
In AI, **representation** means how knowledge is stored, and **mapping** means how real-world facts are connected to symbolic models inside the computer.

1. Representation:

- Refers to the format used to store knowledge.
- Examples include propositional logic, predicate logic, semantic networks, and production rules.
- A good representation should be **expressive, efficient, unambiguous, and support reasoning**.

2. Mapping:

- Defines how objects and events from the real world are mapped into the chosen representation.
- Example: A traffic light system.
    - Real world: {Red, Yellow, Green}.
    - Representation: Propositional variables R, Y, G.
    - Mapping: R = "Red light ON", Y = "Yellow light ON", G = "Green light ON".

Example:
Consider the statement: “All humans are mortal.”

- Representation: Predicate Logic: $\forall x\ (Human(x) \rightarrow Mortal(x))$.
- Mapping: Each person in the real world is mapped to variable x in the logical formula.

Thus, **representation + mapping = meaningful AI knowledge**.

3. What are the different approaches to Knowledge Reepresentation? Explain.

Answer:
Knowledge in AI can be represented using different approaches.

1. Logical Representation

- Uses symbols and rules of logic.
- Example: Predicate logic.
- Advantage: Highly expressive and precise.

2. Semantic Networks

- Represents knowledge as a network of nodes (objects/concepts) and edges (relationships).
- Example: "Cat is-an Animal", "Cat has Legs".
- Advantage: Easy to visualize relationships.

3. Frame-Based Representation

- Uses structures called frames to represent objects with slots and values.
- Example: Frame for "Car" may have slots: {Engine, Wheels, Color}.
- Advantage: Similar to object-oriented models.

4. Production Rules

- IF-THEN rules representing actions or inferences.
- Example: IF raining THEN carry umbrella.
- Advantage: Easy to implement in expert systems.

5. Procedural Representation

- Represents knowledge as procedures or algorithms.
- Example: Cooking recipe as a sequence of steps.
- Advantage: Directly usable for action execution.

Thus, each approach has **strengths and weaknesses**, and in practice, hybrid representations are often used.

4. Explain Proportional Logic in Knowledge Representation.

Answer:
Propositional Logic is the simplest form of logic used in AI where knowledge is represented as propositions (statements that are either true or false).

Key Elements:

1. **Propositions** – Statements like "It is raining" (true/false).
2. **Logical Connectives:**
    - AND ($\land$) → Conjunction
    - OR ($\lor$) → Disjunction

    - NOT (¬) → Negation
    - → → Implication
    - ↔ → Bi-conditional
3. **Truth Tables –** Define how truth values are combined.

Example:

- Statement: "If it rains, then the ground is wet."
- Representation: R → W
- Where R = "It rains", W = "Ground is wet".

Advantages:

- Simple and mathematically precise.
- Good for representing facts.

Limitations:

- Cannot express relations between objects or quantifiers like "all" or "some".

Thus, propositional logic is useful for basic reasoning but insufficient for complex knowledge representation.

5. Explain Resolution in Predicate Logic with example.

Answer:
Resolution is an inference rule used in Predicate Logic (First-Order Logic) for **proving the validity of statements**. It works by contradiction: we negate the goal and try to derive a contradiction using rules.

Steps in Resolution:

1. **Convert statements into clausal form (CNF)**.
2. **Negate the goal statement**.

3. **Apply resolution rule** – combine clauses to eliminate variables.
4. **Derive empty clause (contradiction)** – which means the original goal is true.

Example:

- Knowledge base:
    1. $\forall x\ (Human(x) \rightarrow Mortal(x))$
    2. Human(Socrates)
- Goal: Prove Mortal(Socrates).

Step 1: Convert to CNF

1. $\neg Human(x) \lor Mortal(x)$
2. Human(Socrates)

Step 2: Negate Goal
¬Mortal(Socrates)

Step 3: Apply Resolution

- From (1) and (2): Mortal(Socrates)
- From Mortal(Socrates) and ¬Mortal(Socrates) → Contradiction.

Thus, Mortal(Socrates) is proved true.

Advantages of Resolution:

- Mechanically applicable inference rule.
- Basis for theorem proving in AI.

CO-3

1. Give an overview of Game Playing in AI.

Answer:
Game playing is one of the oldest and most widely studied problems in Artificial Intelligence. It deals with developing systems that can play games against humans or other computers by making intelligent decisions.

Key Features of Game Playing:

1. **Competitive Environment** – Two or more players compete, e.g., chess, tic-tac-toe.
2. **Adversarial Search** – Each player tries to maximize their gain while minimizing the opponent's gain.
3. **State Space Representation** – The game is represented as a search tree with states and moves.
4. **Utility/Evaluation Function** – Used to assign scores to game states.
5. **Optimal Strategy** – AI tries to guarantee the best possible outcome against an opponent.

Types of Games:

- **Perfect Information Games** – Players have complete knowledge of the game state (e.g., Chess, Tic-Tac-Toe).
- **Imperfect Information Games** – Players have partial knowledge (e.g., Poker, Bridge).
- **Deterministic Games** – No element of chance (e.g., Chess).
- **Stochastic Games** – Involves chance (e.g., Backgammon, Ludo).

Example: Chess AI uses adversarial search to calculate the best possible move against the opponent.

Thus, game playing demonstrates how AI can deal with **planning, strategy, and decision-making in adversarial conditions**.

2. Explain MiniMax Search procedure in AI.

Answer:
The **MiniMax algorithm** is a recursive adversarial search method used in two-player games (one MAX player and one MIN player).

Idea:

- The MAX player tries to maximize the score.
- The MIN player tries to minimize the score.
- The algorithm explores the game tree, assigning values to terminal states and propagating them upward until the best decision is found.

Steps in MiniMax:

1. **Generate the game tree** up to a certain depth.
2. **Apply evaluation function** to terminal states (leaf nodes).
3. **Back-propagate values**:
   - At MAX nodes → choose the maximum value among children.
   - At MIN nodes → choose the minimum value among children.
4. **Select the move** that leads to the optimal value for MAX player.

Example (Tic-Tac-Toe):

- If MAX can win in one move, assign value +10.
- If MIN can win, assign value –10.
- If draw, assign 0.
- The MiniMax algorithm ensures the best outcome against an optimal opponent.

Advantages:

- Provides an **optimal strategy** when search tree is fully explored.
- Guarantees the best decision assuming opponent plays optimally.

Limitations:

- Very slow for large games like chess due to exponential growth of game tree.

Thus, MiniMax is a fundamental algorithm for adversarial game playing.

3. What are the Alpha-Beta cutoffs? How do they improve MiniMax?

Answer:
Alpha-Beta pruning is an optimization of the MiniMax algorithm. It reduces the number of nodes explored in the game tree without affecting the final result.

Key Terms:

- **Alpha (α):** Best value that MAX can guarantee so far.
- **Beta (β):** Best value that MIN can guarantee so far.

Working:

1. Traverse the game tree like MiniMax.
2. While evaluating nodes:
   - If MAX finds a value ≥ β → prune (stop exploring) because MIN will never allow it.
   - If MIN finds a value ≤ α → prune because MAX will never allow it.
3. Continue until decision is found with fewer nodes explored.

Example:

- Suppose MIN already has a move with value 3.
- While checking another branch, if MAX gets value ≥ 4, then MIN will never allow that branch → prune it.

Advantages of Alpha-Beta Pruning:

1. Reduces search space drastically.
2. In best case, it cuts computation time by **half**.
3. Makes MiniMax practical for complex games like chess.

Important Note:
The **order of moves** affects pruning efficiency. If best moves are evaluated first, pruning is maximum.

Thus, Alpha-Beta pruning makes game-playing algorithms more efficient while preserving **optimal decision making**.

CO-4

1. Explain the biological inspiration behind Genetic Algorithms.

Answer:
Genetic Algorithms (GAs) are inspired by the process of **natural evolution** and **genetics** in biology. The central idea is that solutions to a problem can evolve over time by applying principles similar to those in biological systems.

Biological Concepts Used in GAs:

1. **Population** – A group of individuals (possible solutions).
2. **Chromosomes** – Representation of solutions, usually as binary strings.
3. **Genes** – Parts of a chromosome representing parameters of the solution.
4. **Fitness Function** – Measures how good an individual is (like survival ability in nature).
5. **Selection** – Fitter individuals are more likely to reproduce.
6. **Crossover (Recombination)** – Combining genes of two parents to form offspring.
7. **Mutation** – Random changes in genes to maintain diversity.

Example from Nature:

- In evolution, species adapt to their environment through natural selection.
- Similarly, in GAs, candidate solutions evolve toward an optimal or near-optimal solution.

Thus, Genetic Algorithms are a computational **simulation of Darwin's "Survival of the Fittest" principle**.

2. Define Genetic Algorithms (GA) and explain their working.

Answer:
A **Genetic Algorithm (GA)** is a search and optimization technique based on the principles of **natural selection and genetics**. It is used to find approximate solutions to complex problems where traditional methods fail.

Steps in Genetic Algorithm:

1. **Initialization** – Generate an initial population of random solutions.
2. **Evaluation** – Calculate fitness of each solution using a fitness function.
3. **Selection** – Choose parents based on fitness (better solutions get higher probability).
4. **Crossover** – Combine selected parents to produce new offspring.
5. **Mutation** – Introduce small random changes to maintain diversity.
6. **Replacement** – Form a new generation with offspring.
7. **Termination** – Repeat until a stopping condition (fixed generations or satisfactory fitness).

Applications of GAs:

- Function optimization.
- Machine learning (feature selection, parameter tuning).
- Scheduling and planning problems.
- Engineering design optimization.

Thus, GAs provide a **robust and adaptive method** for solving optimization problems.

3. Explain the significance of the Selection operator in Genetic Algorithms.

Answer:
The **Selection operator** is one of the most important parts of a Genetic Algorithm. It decides **which individuals from the population will reproduce** to create the next generation.

Role of Selection:

1. Ensures that **fitter individuals** (with higher fitness values) are more likely to survive and reproduce.
2. Mimics **natural selection** – “survival of the fittest.”
3. Controls the **exploration vs. exploitation** balance.

Common Selection Methods:

- **Roulette Wheel Selection (Fitness Proportionate Selection):** Probability of selection is proportional to fitness.
- **Tournament Selection:** Randomly select a few individuals and choose the best among them.
- **Rank Selection:** Individuals ranked by fitness; selection probability depends on rank.

Significance:

- Increases convergence speed toward optimal solution.
- Helps avoid wasting resources on poor solutions.
- Maintains diversity in the population when combined with mutation.

Thus, **selection ensures that good traits are carried forward to future generations**, improving solution quality.

4. Explain the significance of Crossover operator in Genetic Algorithms.

Answer:
The **Crossover operator** (also called recombination) is the primary operator that generates new solutions by combining genetic material from two parent solutions.

Working of Crossover:

- Two parents are chosen.
- A crossover point is selected in the chromosome.
- Parts of parent chromosomes are swapped to produce offspring.

Types of Crossover:

1. **Single-Point Crossover** – A single cut point divides parents, and segments are exchanged.
2. **Two-Point Crossover** – Two cut points; middle segment exchanged.
3. **Uniform Crossover** – Each gene is independently chosen from one of the parents.

Significance:

1. Introduces **new combinations of genetic material** → leads to better offspring.
2. Ensures **information exchange** between solutions.
3. Increases the chance of finding the global optimum.
4. Mimics natural reproduction where offspring inherit traits from both parents.

Example:

- Parent 1: `11001`
- Parent 2: `10111`
- After single-point crossover at position 3 → Offspring 1: `11011`, Offspring 2: `10101`.

Thus, **crossover is the main source of innovation in GAs** that helps explore new areas of the search space.

CO-5

1. Give an introduction to Prolog and its features.

Answer:
Prolog (Programming in Logic) is a high-level logic programming language used mainly in **Artificial Intelligence and Computational Linguistics**. It is based on **first-order predicate logic**.

Key Features of Prolog:

1. **Declarative Language** – Programmer specifies *what to solve*, not *how to solve*.
2. **Facts, Rules, and Queries** – Knowledge is represented using facts and rules; queries are asked to infer results.
3. **Automatic Backtracking** – Prolog automatically searches for solutions by backtracking when a rule fails.
4. **Pattern Matching** – Uses unification to match facts and goals.
5. **Applications** – Natural language processing, expert systems, theorem proving, and knowledge representation.

Thus, Prolog is widely used for problems that involve **symbolic reasoning and non-numeric computation**.

2. How do we convert English sentences in Prolog facts and rules? Give an example.

Answer:
In Prolog, **facts** represent basic information and **rules** represent logical relationships.

Conversion Steps:

1. Identify entities and relationships in English.
2. Express them in Prolog using predicates.
3. Use **facts** for direct statements, and **rules** for conditional statements.

Examples:

- English Fact: *"John is a man."*
  → Prolog: `man(john).`
- English Rule: *"X is mortal if X is a man."*
  → Prolog: `mortal(X) :- man(X).`
- Query: *"Is John mortal?"*
  → Prolog: `?- mortal(john).` → **Yes**

Thus, English sentences are transformed into **logical facts and rules** for reasoning in Prolog.

3. Explain Goals in Prolog with an example.

Answer:
A **goal** is a query given to the Prolog system to check whether it can be satisfied using available facts and rules.

How Goals Work:

- Written as a predicate (like a fact).
- Prolog tries to prove the goal by searching its knowledge base.
- If it succeeds, it returns **true** (Yes); otherwise, **false** (No).

  Example:

  father(john, david).

  father(david, peter).

  grandfather(X, Y) :- father(X, Z), father(Z, Y).

- Query/Goal: `?- grandfather(john, peter).`
- Prolog finds: `john → david → peter` → returns **Yes**.

Thus, goals are **questions asked to Prolog**, and the system checks whether they follow from facts and rules.

4. Explain important Prolog terminology.

Answer:
Some important **Prolog terms** are:

1. **Atom** – A constant symbol (e.g., `john`, `apple`).
2. **Number** – Numeric constants (e.g., `12`, `3.14`).
3. **Variable** – Identifiers starting with uppercase (e.g., `X`, `Person`).

4. **Predicate** – Represents a relation (e.g., `father(john, peter)`).
5. **Fact** – A basic statement that is always true (e.g., `man(john).`).
6. **Rule** – A logical implication (e.g., `mortal(X) :- man(X).`).
7. **Goal/Query** – A question asked (e.g., `?- man(john).`).
8. **Clause** – A fact or rule.

Thus, these terms define the **building blocks of Prolog programs**.

5. Explain the use of Variables in Prolog with examples.

Answer:
In Prolog, **variables** are placeholders that can take any value.

Key Points:

- Written with an uppercase letter or underscore.
- Used for **pattern matching** and **generalization**.

Example:

parent(john, david).

parent(david, peter).

- Query: `?- parent(john, X).`
- Answer: `X = david.`
- Query: `?- parent(X, peter).`
- Answer: `X = david.`

Thus, variables allow **general queries** without specifying exact values.

6. Explain control-structures in Prolog.

Answer:
Unlike procedural languages, Prolog's control structures are based on **backtracking and search**.

Control Mechanisms:

1. **Conjunction (`,`)** – Both goals must be true.
   - Example: `?- man(john), mortal(john).`
2. **Disjunction (`;`)** – At least one goal must be true.
   - Example: `?- man(john); woman(john).`
3. **Negation (`\+`)** – Represents "not provable."
   - Example: `?- \+ man(mary).`
4. **Cut (`!`)** – Prevents backtracking beyond a point.
   - Example: Used in optimization of rules.

Thus, Prolog controls the flow of reasoning using **logical operators and backtracking**.

7. Explain arithmetic operators in Prolog.

Answer:
Prolog supports arithmetic operations for numerical calculations.

Common Arithmetic Operators:

- Addition: `+`
- Subtraction: `-`
- Multiplication: `*`
- Division: `/`
- Modulus: `mod`

- Exponent: `**`

Usage Example:

sum(A, B, Result) :- Result is A + B.

- Query: `?- sum(5, 3, X).`
- Answer: `X = 8.`

Note: Arithmetic is done using `is` operator because Prolog treats expressions symbolically unless evaluated.

Thus, arithmetic operators extend Prolog’s reasoning with **numeric computations**.