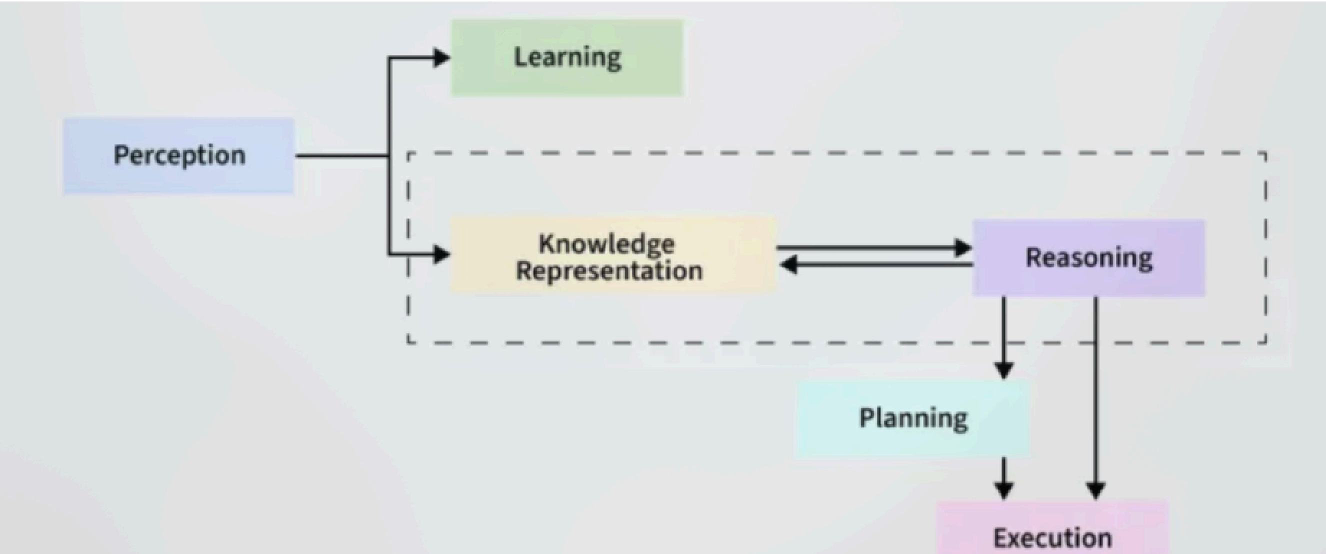
AI UNIT - 2 CONCISE

UNIT 2: Knowledge Representation and Reasoning

1. Knowledge Representation (KR) in AI `[PYQ]`

- **Definition:** Encoding information about the world into formats AI systems can utilize to solve complex tasks.
- Enables machines to reason, learn, and make decisions by structuring data like human understanding. `[FIG]` (Perception -> KR -> Reasoning -> Planning -> Execution -> Learning loop)



- Raw data alone ≠ intelligence; AI must transform data into **structured knowledge**.
- KR defines formats and methods for organizing information.

2. The Synergy of Knowledge and Intelligence

- Symbiotic relationship:
  - **Knowledge as Foundation:** Provides facts, rules, data (e.g., traffic laws for self-driving cars). Intelligence lacks raw material without it.
  - **Intelligence as Application:** Applies knowledge to solve problems (e.g., robot using physics to navigate).
  - **Interdependence:** Static knowledge can become obsolete without adaptive intelligence. Intelligence without knowledge cannot reason/learn (e.g., AI without medical DB can't diagnose).
  - **Synergy:** Effective AI merges robust knowledge bases (the *what*) with reasoning algorithms (the *how*). (e.g., ChatGPT: language data + transformer models).

3. Core Methods of Knowledge Representation `[PYQ]`

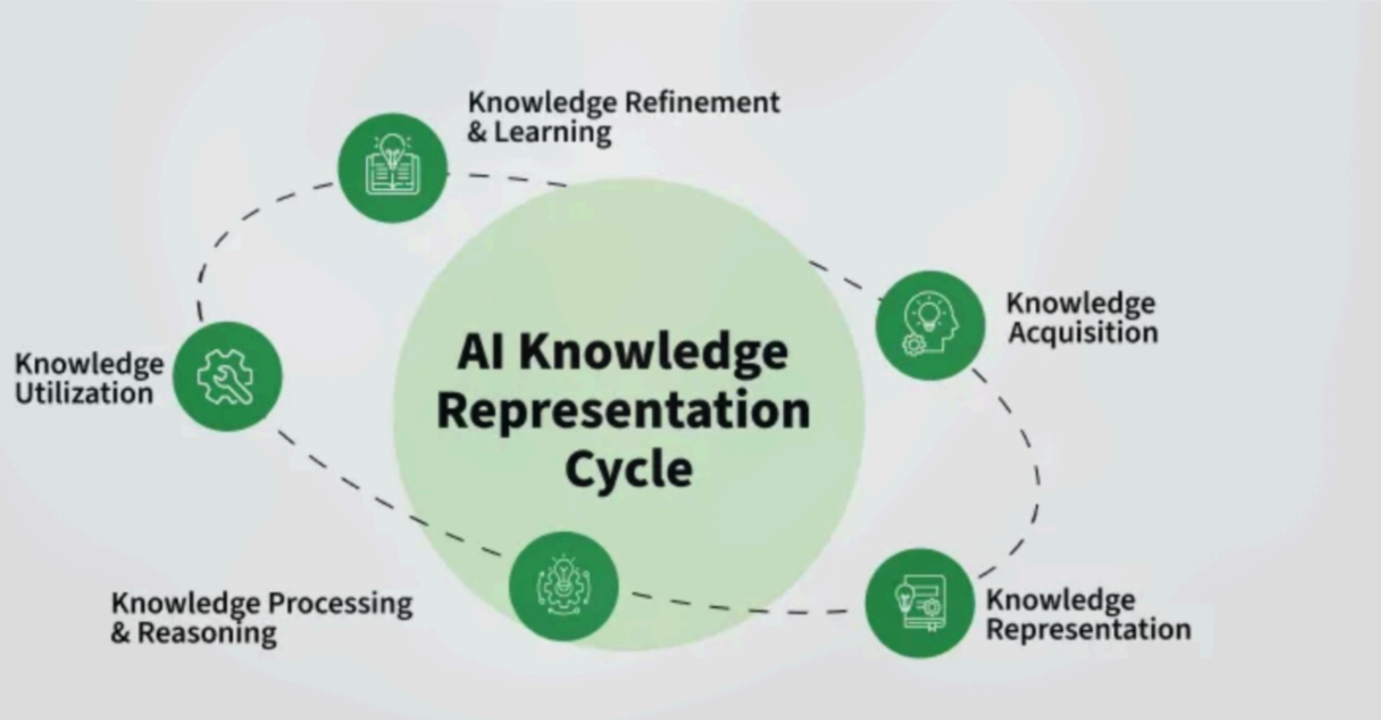
- **A. Logic-Based Systems:** `[PYQ]` Use formal rules; prioritize precision; ideal for deterministic environments.

- **Propositional Logic (PL):** `[PYQ]`
  - Represents knowledge as declarative statements (propositions) linked by logical operators (AND, OR, NOT).
  - Format: "IF condition THEN conclusion."
  - Simple, but struggles with complex relationships.
- **First-Order Logic (FOL) / Predicate Logic:** `[PYQ]`
  - Extends PL with variables, quantifiers (∀, ∃), predicates.
  - Expresses statements like "∀x Mortal(x)". Supports nuanced reasoning.
  - Demands significant computational resources.

- **B. Structured Representations:** Organize knowledge hierarchically or via networks.
  - **Semantic Networks:** `[PYQ]` `[FIG]`
    - Nodes (concepts) and edges (relationships, e.g., "Is-A", "Has-Part").
    - Example: "Dog" --(Is-A)--> "Animal".
    - Simplify inheritance reasoning; may lack formal semantics.
  - **Frames:** `[PYQ]` `[FIG]`
    - Group related attributes (slots) into structured "frames" (e.g., "Vehicle" frame with slots: wheels, engine).
    - Slots can have facets (constraints, default values, procedures).
    - Excel in default reasoning; struggle with exceptions. Supports inheritance.
  - **Ontologies:** `[PYQ]`
    - Define concepts, hierarchies, relationships within a domain using standards (e.g., OWL).
    - Power semantic search, healthcare diagnostics by standardizing terminology.
- **C. Probabilistic Models:** Handle uncertainty by assigning probabilities to outcomes.
  - **Bayesian Networks:** `[PYQ]` `[FIG]`
    - Directed Acyclic Graphs (DAGs) to model causal relationships.
    - Nodes: variables; Edges: conditional dependencies.
    - Predict likelihood (e.g., equipment failure based on history).
  - **Markov Decision Processes (MDPs):** `[FIG]`
    - Model sequential decision-making in dynamic environments.
    - Help robotics evaluate potential actions and rewards.
- **D. Distributed Representations:** Leverage neural networks to encode knowledge as numerical vectors (embeddings).
  - **Embeddings:** Convert words, images, entities into dense vectors (e.g., Word2Vec for semantic analysis).

- **Knowledge Graphs:** [PYQ] [FIG]
  - Combine graph structures with embeddings to represent entities and relationships.
  - Enhance search by linking related concepts (e.g., Google's Knowledge Graph).

4. The AI Knowledge Cycle [PYQ] [FIG]



- Continuous process: AI systems acquire, process, utilize, and refine knowledge.
  - **1. Knowledge Acquisition:** Gather data (structured DBs, text, images, interactions) via ML, NLP, CV.
  - **2. Knowledge Representation:** Structure acquired knowledge for efficient storage/retrieval using methods above.
  - **3. Knowledge Processing & Reasoning:** Apply logical inference, probabilistic models, deep learning to:
    - Draw conclusions (deductive, inductive).
    - Solve problems (heuristic search, optimization).
    - Adapt (reinforcement learning).
  - **4. Knowledge Utilization:** Apply knowledge to real-world tasks (decision-making, predictions, automation). E.g., virtual assistants, recommendation systems, self-driving cars.
  - **5. Knowledge Refinement & Learning:** Continuously update knowledge base via feedback loops (reinforcement learning, supervised fine-tuning, active learning). Ensures AI evolves.

5. Types of Knowledge in AI [PYQ] [FIG] (Knowledge types diagram)

- **A. Declarative Knowledge (Descriptive / "What"):** [PYQ]
  - Facts and information about the world. Stored in DBs, ontologies, KGs.

    - E.g., "Paris is the capital of France." Used by search engines, virtual assistants.
- **B. Procedural Knowledge (Imperative / "How-To"):** `[PYQ]`
    - Steps/methods to perform tasks. Encoded in rule-based systems, decision trees, ML models.
    - E.g., How to solve a quadratic equation. Used by expert systems, robotics.
- **C. Meta-Knowledge ("Knowledge About Knowledge"):** `[PYQ]`
    - Knowledge about how information is structured, used, validated.
    - Helps determine reliability, relevance, applicability. Crucial for filtering misinformation.
- **D. Heuristic Knowledge ("Experience-Based"):** `[PYQ]`
    - Derived from experience, intuition, trial-and-error. Allows educated guesses when exact answers are hard.
    - E.g., Navigation system suggesting alternate route based on past traffic. Used in A* search.
- **E. Common-Sense Knowledge:** `[PYQ]`
    - Basic understanding of the world humans acquire naturally. Challenging for AI.
    - E.g., "Water is wet," "If you drop something, it will fall." Requires contextual understanding.
    - Researchers use KBs like ConceptNet.
- **F. Domain-Specific Knowledge:**
    - Specialized knowledge for fields like medicine, finance, law. Detailed and structured.
    - E.g., Medical AI uses knowledge of symptoms, diseases.
- **G. Structural Knowledge:**
    - Basic knowledge for problem-solving; describes relationships between concepts (kind-of, part-of, grouping).

6. What to Represent in AI Systems:

- **Objects:** Facts about objects (e.g., Guitars have strings).
- **Events:** Actions occurring in the world.
- **Performance:** Knowledge about how to do things.
- **Meta-knowledge:** Knowledge about what we know.
- **Facts:** Truths about the real world.
- **Knowledge-Base (KB):** Central component of knowledge-based agents; a group of sentences (formal technical term). `[PYQ]`

7. Approaches to Knowledge Representation (Categorization)

- **A. Simple Relational Knowledge:** `[FIG]`
    - Uses relational methods (tables: rows/columns) to store facts. Simplest type.
    - Used in database systems. Low opportunity for inference.

- **B. Inheritable Knowledge:** `[PYQ]` `[FIG]`
  - Knowledge acquired through learning, transferable/inheritable by other AI systems.
  - Data stored in a hierarchy of classes. Boxed nodes (objects/values), arrows (pointers).
  - Allows faster learning, knowledge transfer across domains.
- **C. Inferential Knowledge:** `[PYQ]`
  - Ability to draw logical conclusions or make predictions from available data.
  - Represents knowledge in formal logic.
  - Example: `Footballer(Alex)`, `∀x (Footballer(x) → Athlete(x))`, therefore `Athlete(Alex)`.
- **D. Procedural Knowledge:** (Covered in "Types of Knowledge")
  - Knowledge/instructions to perform a specific task. Often uses IF-THEN rules.

8. Techniques of Knowledge Representation (Implementation Methods) `[PYQ]` `[FIG]`

- **A. Logical Representation:** `[PYQ]`
  - Language with definite rules, deals with propositions, no ambiguity.
  - **Syntax:** Rules for constructing legal sentences.
  - **Semantics:** Rules for interpreting sentences, assigning meaning.
  - Categorized into: Propositional Logic, Predicate Logic.
  - Advantages: Performs logical reasoning, basis for programming languages.
  - Disadvantages: Restrictions, challenging to work with, may not be natural/efficient for inference.
- **B. Semantic Network Representation:** (Covered in "Core Methods") `[PYQ]` `[FIG]`
- **C. Frame Representation:** (Covered in "Core Methods") `[PYQ]` `[FIG]`
- **D. Production Rules:** (Covered in Unit 1 - Production Systems) `[PYQ]` `[FIG]`

9. Propositional Logic (PL) / Statement Logic `[PYQ]`

- Simplest form of logic; statements are propositions (declarative, either true or false).
- Also called Boolean logic (works on True/False or 0/1).
- **Syntax:**
  - **Atomic Propositions:** Simple propositions, single symbol (e.g., P, "It is Sunday").
  - **Compound Propositions:** Constructed by combining simpler propositions using logical connectives.

- **Logical Connectives:** [PYQ] [FIG] (Truth Table for each)

For Implication:

P	Q	P→ Q
True	True	**True**
True	False	**False**
False	True	**True**
False	False	**True**

For Biconditional:

P	Q	P⇔ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**True**

  - **Negation (¬, NOT):** ¬P.
  - **Conjunction (∧ , AND):** P ∧ Q (e.g., "Rohan is intelligent AND hardworking").
  - **Disjunction (∨ , OR):** P ∨ Q (e.g., "Ritika is a doctor OR engineer").
  - **Implication (→, IF...THEN):** P → Q (e.g., "IF it is raining THEN street is wet").
  - **Biconditional (↔, IF AND ONLY IF):** P ↔ Q (e.g., "IF I am breathing THEN I am alive").
- **Truth Tables:** [PYQ] [FIG] Tabular representation of truth values for all possible combinations of propositions and connectives.
- **Precedence of Connectives:** Parenthesis > Negation > Conjunction > Disjunction > Implication > Biconditional.
- **Key Terms:**
  - **Tautology:** A proposition formula always true.
  - **Contradiction:** A proposition formula always false.
  - **Contingency:** A formula that can be true or false.
- **Limitations of Propositional Logic:** [PYQ]
  - **Limited expressivity:** Cannot represent complex relationships between objects/concepts easily.
  - **Inability to handle quantifiers:** Cannot represent "all" or "some" concisely.
  - **Lack of support for negation (nuanced):** Difficult to represent complex negative statements.

10. Predicate Logic / First-Order Logic (FOL) [PYQ]

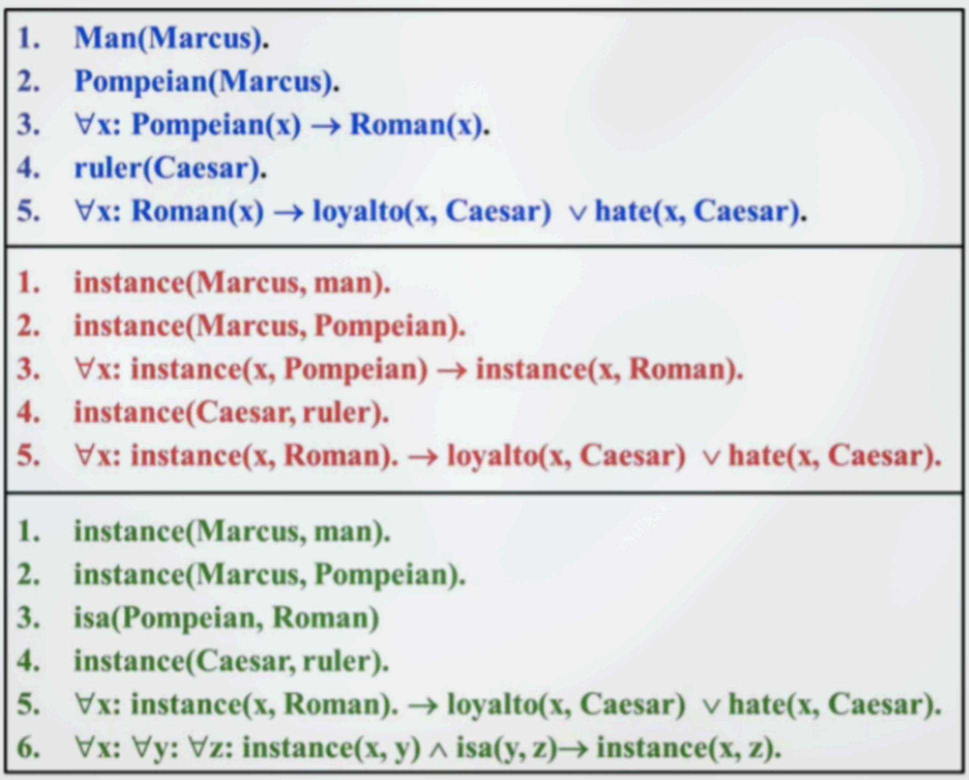
- Extends PL; more expressive. Represents objects, properties, relations, quantifiers.
- Also known as First Order Predicate Calculus Logic (FOPL).
- **Basic Components / Elements of FOL Syntax:** [PYQ]
  - **Constants:** Specific objects (e.g., John, Mumbai, 2).
  - **Variables:** Stand for unspecified objects (e.g., x, y, z).

- **Predicates:** Properties of objects or relations between objects; return true/false (e.g., `IsHungry(x)`, `Likes(Alice, Bob)`).
- **Functions:** Map objects to other objects (e.g., `MotherOf(x)`).
- **Connectives:** ∧ , ∨ , ¬, →, ↔ (same as PL).
- **Equality (=):** Asserts two terms refer to the same object.
- **Quantifiers:** Specify scope of variables. `[PYQ]`
  - **Universal Quantifier (∀, "for all"):** Statement is true for all objects in the domain. (e.g., `∀x Human(x) → Mortal(x)`). Uses implication (→).
  - **Existential Quantifier (∃, "there exists"):** Statement is true for at least one object. (e.g., `∃x Boy(x) ∧ Intelligent(x)`). Uses conjunction (∧).

- **Syntax of FOL:**
  - **Terms:** Refer to objects (constants, variables, functions applied to terms).
  - **Atomic Sentences:** Predicate symbol followed by a parenthesized list of terms (e.g., `Brothers(Ravi, Ajay)`).
  - **Complex Sentences:** Formed by combining atomic sentences with connectives and quantifiers.

- **Free and Bound Variables:**
  - **Free Variable:** Occurs outside the scope of its quantifier (e.g., z in `∀x ∃y P(x,y,z)`).
  - **Bound Variable:** Occurs within the scope of its quantifier (e.g., x, y in `∀x ∃y P(x,y,z)`).

- **Representing INSTANCE and ISA Relationships:** `[PYQ]` `[FIG]`



1. Man(Marcus).
2. Pompeian(Marcus).
3. $\forall$x: Pompeian(x) $\rightarrow$ Roman(x).
4. ruler(Caesar).
5. $\forall$x: Roman(x) $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).

1. instance(Marcus, man).
2. instance(Marcus, Pompeian).
3. $\forall$x: instance(x, Pompeian) $\rightarrow$ instance(x, Roman).
4. instance(Caesar, ruler).
5. $\forall$x: instance(x, Roman). $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).

1. instance(Marcus, man).
2. instance(Marcus, Pompeian).
3. isa(Pompeian, Roman)
4. instance(Caesar, ruler).
5. $\forall$x: instance(x, Roman). $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).
6. $\forall$x: $\forall$y: $\forall$z: instance(x, y) $\wedge$ isa(y, z)$\rightarrow$ instance(x, z).

Figure: Three ways of representing class membership

  - **Instance:** Class membership (e.g., `Man(Marcus)` or `instance(Marcus, Man)`).

- **ISA:** Class inclusion/subclass (e.g., `∀x Pompeian(x) → Roman(x)` or `isa(Pompeian, Roman)`).
- Requires axioms for `isa` (e.g., `∀x ∀y ∀z instance(x,y) ∧ isa(y,z) → instance(x,z)`).

11. Rules of Inference in Logic `[PYQ]` `[FIG]`

- Logical principles to derive conclusions from existing information.
- **Types of Inference Rules:**
  - **Modus Ponens:** If (P → Q) and P are true, then Q is true. `[PYQ]`
    - `((P → Q) ∧ P) ⇒ Q`
  - **Modus Tollens:** If (P → Q) is true and Q is false, then P is false. `[PYQ]`
    - `((P → Q) ∧ ¬Q) ⇒ ¬P`
  - **Hypothetical Syllogism:** If (P → Q) and (Q → R) are true, then (P → R) is true. `[PYQ]`
    - `((P → Q) ∧ (Q → R)) ⇒ (P → R)`
  - **Disjunctive Syllogism:** If (P ∨ Q) is true and P is false, then Q is true. `[PYQ]`
    - `((P ∨ Q) ∧ ¬P) ⇒ Q`
  - **Addition:** If P is true, then (P ∨ Q) is true.
    - `P ⇒ (P ∨ Q)`
  - **Simplification:** If (P ∧ Q) is true, then P is true.
    - `(P ∧ Q) ⇒ P`
  - **Resolution:** `[PYQ]` If (P ∨ Q) and (¬P ∨ R) are true, then (Q ∨ R) is true. Fundamental for automated theorem proving.
    - `((P ∨ Q) ∧ (¬P ∨ R)) ⇒ (Q ∨ R)`
- **Applications of Inference:** NLP, Expert Systems, Robotics, Computer Vision.

12. Unification and Resolution in FOL `[PYQ]`

- **Unification:** `[PYQ]` `[FIG]`
  - Process of finding a substitution (assignment of terms to variables) that makes two logical atomic expressions identical (match).
  - Takes two literals, makes them identical. `UNIFY(Ψ1, Ψ2)`.
  - Substitution denoted by θ.
  - **Most General Unifier (MGU):** The substitution that makes expressions identical and is "least constrained."
  - **Conditions for Unification:**
    - Predicate symbols must be the same.

        - Number of arguments must be identical.
        - Fails if a variable needs to be unified with a term containing that same variable (occurs check).
    - **Unification Algorithm:** A recursive algorithm to find MGU.
- **Resolution in FOL:** `[PYQ]` `[FIG]` (Resolution Graph)
    - Theorem proving technique; proves by contradiction (refutation).
    - Operates on clauses in **Conjunctive Normal Form (CNF)**.
        - **Clause:** A disjunction of literals (e.g., `¬P ∨ Q ∨ R`).
        - **CNF:** A conjunction of clauses (e.g., `(A ∨ B) ∧ (¬B ∨ C)`).
    - **Resolution Inference Rule (for FOL):**
        - Given two clauses `L1 ∨ ... ∨ Lk` and `M1 ∨ ... ∨ Mn` containing complementary literals `Li` and `Mj` (i.e., `Li = ¬Mj` after unification with MGU θ).
        - The resolvent is `(L1 ∨ ... ∨ Li-1 ∨ Li+1 ∨ ... ∨ Lk ∨ M1 ∨ ... ∨ Mj-1 ∨ Mj+1 ∨ ... ∨ Mn)θ`.
    - **Steps for Resolution Proof:**
        1. Convert facts/premises into FOL.
        2. Convert FOL statements into CNF:
            - Eliminate implications (→, ↔).
            - Move negations (¬) inwards (De Morgan's laws).
            - Standardize variables apart (rename variables so each quantifier has a unique variable).
            - Skolemize: Eliminate existential quantifiers (replace ∃x P(x) with P(SkolemConstant); replace ∀y∃x P(x,y) with ∀y P(SkolemFunction(y),y)).
            - Drop universal quantifiers (all variables assumed universally quantified).
            - Distribute ∧ over ∨ to get conjunction of disjunctions (clauses).
        3. Negate the statement to be proved (the query/goal). Convert it to CNF and add its clauses to the KB.
        4. Repeatedly apply resolution rule to derive new clauses until an empty clause (contradiction, ⊥) is derived, or no more progress can be made.
        5. If empty clause is derived, original (non-negated) query is true.
    - Example: "John likes peanuts" proof. `[FIG]`

13. Frames in AI: Knowledge Representation and Inheritance `[PYQ]` `[FIG]`

- Data structures to represent stereotypical situations or scenarios (objects, events).
- Introduced by Minsky (1974).
- **Key Components of a Frame:**

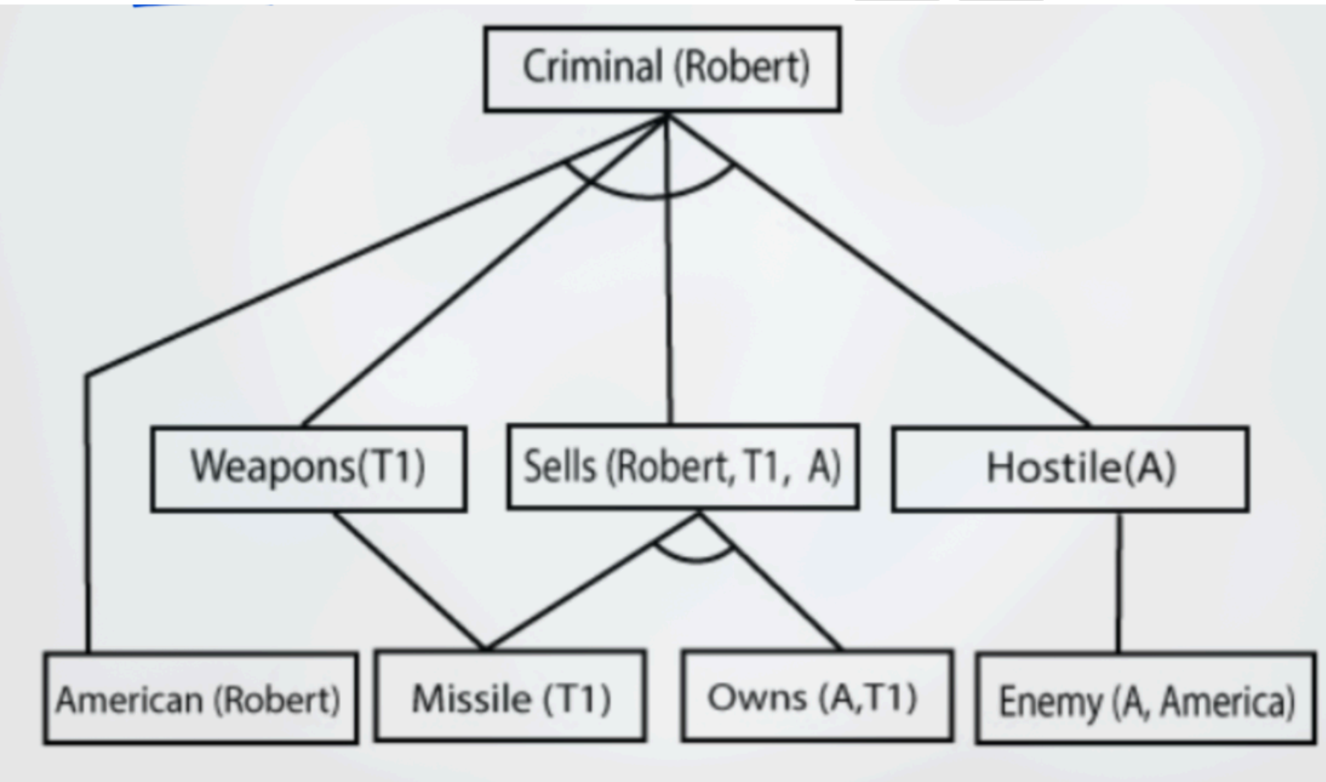
- **Frame Name:** Identifies the concept (e.g., "Book", "Person").
- **Slots:** Attributes or properties of the frame (e.g., for "Person": Name, Age, Occupation). `[PYQ]`
- **Facets:** Additional details or constraints for slots (e.g., for "Age" slot: Type=Integer, Range=0-120, Default Value=30). `[PYQ]`
- **Default Values:** Predefined values for slots if not specified.
- **Procedures (Demons/Methods):** Functions associated with frames.
  - *IF-NEEDED*: Executed when a slot value is required but not present.
  - *IF-ADDED*: Executed when a value is added to a slot.
  - *IF-REMOVED*: Executed when a value is removed from a slot.

- **Frame Inheritance:** `[PYQ]` `[FIG]`
  - Allows a child frame to inherit attributes/properties from a parent frame, creating a hierarchy.
  - **Parent Frame:** Frame from which attributes are inherited.
  - **Child Frame:** Frame that inherits; can add new attributes or **override** inherited ones.
  - **Inheritance Hierarchy:** Tree-like structure of frames.
  - **Extension:** Adding new attributes in a child frame.
- **Applications of Frames:** NLP (context understanding), Expert Systems (domain knowledge), Robotics (object properties), Cognitive Modeling.
- **Advantages:** Organized knowledge, flexibility, reusability.
- **Challenges:** Complexity, context sensitivity, scalability.
- **Frames vs. Ontologies:** `[PYQ]`
  - **Frames:** Represent specific, context-dependent scenarios; less formal; flexible.
  - **Ontologies:** Formal, standardized representation of knowledge across domains; ensure consistency; use languages like OWL.

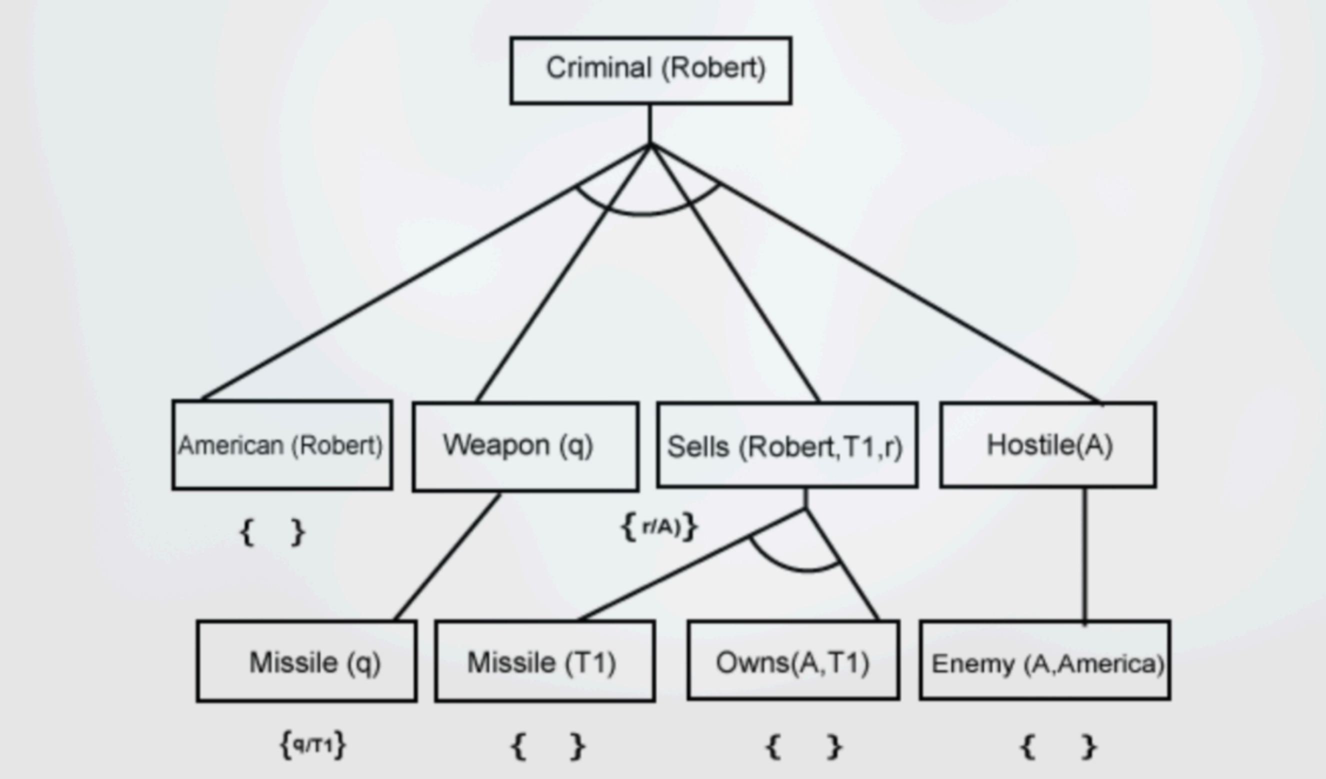
14. Forward and Backward Chaining `[PYQ]` `[FIG]`

- Inference engine modes for rule-based systems (often using definite clauses).
- **Horn Clause:** A clause (disjunction of literals) with *at most one* positive literal.
- **Definite Clause:** A Horn clause with *exactly one* positive literal. (e.g., `(¬P ∨ ¬Q ∨ R)` which is `(P ∧ Q) → R`).

- **A. Forward Chaining (Data-Driven Reasoning / Bottom-Up):** `[PYQ]` `[FIG]`



  - Starts with known facts (atomic sentences in KB).
  - Applies inference rules (Modus Ponens) in the forward direction.
  - If a rule's premises are satisfied by known facts, its conclusion is added to the KB as a new fact.
  - Process repeats until the goal is reached or no new facts can be derived.
  - **Properties:** Down-up approach, data-driven.
  - **Applications:** Planning, design, monitoring, diagnosis, classification.
  - Uses Breadth-First Search strategy.

- **B. Backward Chaining (Goal-Driven Reasoning / Top-Down):** `[PYQ]` `[FIG]`
  

  - Starts with the goal to be proven.
  - Finds rules whose conclusion matches the current goal (or sub-goal).
  - The premises of such rules become new sub-goals.
  - Works backward until all sub-goals are satisfied by known facts in the KB.
  - **Properties:** Top-down approach, goal-driven.
  - **Applications:** Classification, diagnosis, question answering.
  - Uses Depth-First Search strategy.
- **Difference between Forward and Backward Chaining:** `[PYQ]`
  - **Starting Point:** FC (facts) vs. BC (goal).
  - **Direction:** FC (premises to conclusion) vs. BC (conclusion to premises).
  - **Driven by:** FC (data) vs. BC (goal).
  - **Search Strategy:** FC (often BFS-like) vs. BC (often DFS-like).
  - **Efficiency:** FC can be exhaustive; BC can be more focused if goal is specific.

15. Reasoning in Artificial Intelligence `[PYQ]`

- Mental process of deriving logical conclusions and making predictions from available knowledge, facts, beliefs. Inferring facts from existing data.
- **Types of Reasoning:** `[PYQ]`
  - **A. Deductive Reasoning (Top-Down):** `[PYQ]` `[FIG]`
    - Deducing new information from logically related known information.
    - Argument's conclusion *must be true* if premises are true. (Valid reasoning).

        - General premises → Specific conclusion.
        - Example: Premise 1: All humans eat veggies. Premise 2: Suresh is human. Conclusion: Suresh eats veggies.
    - **B. Inductive Reasoning (Bottom-Up):** `[PYQ]` `[FIG]`
        - Arriving at a conclusion (generalization) using a limited set of specific facts/observations.
        - Premises provide *probable support* for the conclusion; truth of premises does not guarantee truth of conclusion.
        - Specific facts/data → General statement.
        - Example: Premise: All pigeons seen in the zoo are white. Conclusion: All pigeons are white.
    - **C. Abductive Reasoning:** `[PYQ]`
        - Finding the *most likely explanation* or conclusion for single/multiple observations.
        - Form of logical inference; premises do not guarantee the conclusion.
        - Example: Implication: Cricket ground is wet IF it is raining. Axiom: Cricket ground is wet. Conclusion: It IS raining (most likely).
    - **D. Common Sense Reasoning:** `[PYQ]`
        - Informal reasoning gained through experiences. Simulates human presumptions about everyday events.
        - Relies on good judgment, heuristic knowledge/rules.
        - Example: "One person can be at one place at a time."
    - **E. Monotonic Reasoning:** `[PYQ]`
        - Once a conclusion is taken, it remains the same even if new information is added to KB.
        - Adding knowledge *does not decrease* the set of propositions that can be derived.
        - Used in conventional reasoning systems, logic-based systems. Theorem proving is an example.
        - Advantages: Old proofs remain valid.
        - Disadvantages: Cannot represent real-world scenarios (facts change), cannot express hypothesis knowledge easily.
    - **F. Non-monotonic Reasoning:** `[PYQ]`
        - Some conclusions may be invalidated if more information is added to KB.
        - Deals with incomplete and uncertain models.
        - Example: KB: {Birds can fly, Penguins cannot fly, Pitty is a bird}. Conclusion: Pitty can fly. Add: "Pitty is a penguin." New Conclusion: Pitty cannot fly (invalidates previous).
        - Advantages: Useful for real-world systems (robot navigation), can use probabilistic facts/make assumptions.
        - Disadvantages: Old facts may be invalidated, not used for theorem proving in the classical sense.

16. Probabilistic Reasoning & Uncertainty `[PYQ]`

- **Uncertainty:** Situations where we are not sure about the truth of predicates or statements.
- **Causes of Uncertainty:** Unreliable sources, experimental errors, equipment fault, temperature variation, climate change.
- **Probabilistic Reasoning:** `[PYQ]`
  - Knowledge representation applying probability theory to indicate uncertainty in knowledge.
  - Combines probability theory with logic.
  - Handles uncertainty due to "laziness" (not modeling all factors) or "ignorance" (lack of complete knowledge).
- **Need for Probabilistic Reasoning in AI:**
  - When there are unpredictable outcomes.
  - When predicate specifications/possibilities become too large.
  - When unknown errors occur during experiments.
- **Methods to Solve Problems with Uncertain Knowledge:**
  - **Bayes' Rule / Bayesian Statistics:** `[PYQ]` (Often covered in more detail later).
    - `P(H|E) = [P(E|H) * P(H)] / P(E)`
    - P(H|E): Posterior probability of hypothesis H given evidence E.
    - P(E|H): Likelihood of evidence E given hypothesis H.
    - P(H): Prior probability of hypothesis H.
    - P(E): Probability of evidence E.