UNIT – IV

Knowledge Representation: Ontological Engineering, Categories and Objects, Events. Mental Events and Mental Objects, Reasoning Systems for Categories, Reasoning with Default Information.

Classical Planning: Definition of Classical Planning, Algorithms for Planning with State-Space Search, Planning Graphs, other Classical Planning Approaches, analysis of Planning approaches.

Knowledge Representation:

Knowledge Representation (KR) is a field of Artificial Intelligence (AI) that focuses on how to formally represent information such that an AI system can use it to solve complex problems

Ontological Engineering

- ✶ **Definition:** Ontological engineering is the process of building ontologies. An ontology, in the context of AI and computer science, is a formal specification of a shared conceptualization. It defines a set of concepts and categories (classes) that are used to describe a domain of knowledge, along with the relationships between these concepts.
- ✶ **Purpose:** The goal of ontological engineering is to create a structured and unambiguous representation of knowledge that can be understood and processed by both humans and machines. It provides a common vocabulary and a hierarchical structure for organizing information.

✶ Key Components of an Ontology:

- **Classes/Concepts:** Broad categories of things (e.g., "Person," "City," "Disease," "Vehicle").
- **Instances/Individuals:** Specific examples of classes (e.g., "Dr. Smith" is an instance of "Person," "Paris" is an instance of "City").
- **Attributes/Properties:** Characteristics of classes or instances (e.g., a "Person" has an "age," a "City" has a "population").
- **Relations/Relationships:** How concepts are connected (e.g., "is-a" (subclass-of), "part-of," "located-in," "has-owner").
- **Axioms:** Logical statements that define the meaning of terms and constrain the interpretation of the ontology

- ✶ **Process:** Typically involves stages like requirements gathering, conceptualization, formalization, implementation, and maintenance.
- ✶ **Examples:** Gene Ontology (GO) for biology, schema.org for web content, various domain-specific ontologies in medicine, engineering, etc.
- ✶ **Tools:** Ontology editors like Protégé, TopBraid Composer

Categories and Objects

- **Definition:** This refers to the fundamental way we organize knowledge by classifying entities into categories (or classes) and identifying individual instances **(objects)** within those categories.
- **Categories (Classes):** Represent sets of objects that share common characteristics. **They allow for generalization and abstraction**. **For example,** "Mammal" is a category that includes all animals that share certain features (warm-blooded, fur, milk production).
- **Objects (Instances):** Refer to specific, individual entities. "My pet dog, Fido," is an object that belongs to the category "Dog," which in turn belongs to "Mammal."
- **Relationship:** The primary relationship between categories and objects is "instance-of" (an object is an instance of a category) and "subclass-of" (one category is a more specific type of another category, e.g., "Dog" is a subclass of "Mammal").
- **Importance:** This categorization allows for:
    - **Inheritance:** Properties defined for a category can be inherited by its subclasses and instances (e.g., if "Mammals" breathe air, then "Dogs" breathe air, and Fido breathes air).
    - **Efficient Reasoning:** Allows systems to make inferences about objects based on their category membership.
    - **Knowledge Organization:** Provides a structured way to store and retrieve information.

Events

- **Definition:** Events are occurrences or happenings that take place over a period of time. They are crucial for representing **dynamic aspects of the world, actions, processes, and changes.**
- **Representation Challenges:** Representing events effectively requires capturing:
    - **Participants/Roles:** Who or what is involved (agents, patients, instruments).
    - **Time:** When did it happen (start time, end time, duration, temporal relations to other events).
    - **Location:** Where did it happen.
    - **Preconditions:** What must be true for the event to occur.
    - **Effects/Postconditions:** What changes result from the event.
- **Approaches to Event Representation:**

    - **Situational Calculus:** A logical framework where actions transform states of the world.
    - **Event Calculus:** Another logical framework focused on the effects of events over time.
    - **Semantic Networks/Frames:** Events can be represented as nodes or frames with slots for participants, time, etc.
    - **Ontologies:** Events can be defined as classes in an ontology, with properties describing their attributes.
- **Examples:** "John ate an apple," "The meeting started at 10 AM," "The car crashed into the tree."

Mental Events and Mental Objects

- **Definition:** This is a more complex and philosophical area within KR, delving into how to represent internal states, beliefs, desires, intentions, knowledge, and other cognitive phenomena of agents (both human and artificial).
- **Mental Events:** These are processes that occur within an agent's mind.
    - **Examples:** "Believing," "Knowing," "Deciding," "Perceiving," "Recalling," "Thinking."
- **Mental Objects:** These are the contents or targets of mental events.
    - **Examples:** "A belief that the sky is blue," "The knowledge that 2+2=4," "The desire to eat pizza," "The perception of a red car."
- **Challenges:**
    - **Opacity:** Mental states are often "opaque" to direct observation.
    - **Intensionality:** The meaning of a mental object can depend on the way it's represented (e.g., "the morning star" vs. "the evening star" both refer to Venus, but a person might believe one is true without believing the other).
    - **Context Dependence:** Beliefs and desires can change with context.
    - **Self-Reference:** Agents can have beliefs about their own beliefs.
- **Approaches:**
    - **Modal Logics:** Logics like epistemic logic (for knowledge) and doxastic logic (for belief) introduce modal operators (e.g., $K_a\phi$ means "agent a knows that ϕ").
    - **Belief-Desire-Intention (BDI) Architectures:** A popular model in agent systems that explicitly represents and reasons about an agent's beliefs, desires, and intentions.
    - **Folk Psychology:** Attempting to formalize common-sense notions of mental states.

Reasoning Systems for Categories

- **Definition:** These are the inference mechanisms and logical frameworks used to draw conclusions based on knowledge represented in categories and objects. They leverage the hierarchical structure and properties of categories.
- **Key Reasoning Tasks:**
    - **Classification:** Determining which category an object belongs to (e.g., "Is Fido a dog?").

    - **Subsumption:** Determining if one category is a subcategory of another (e.g., "Is Dog a subclass of Mammal?").
    - **Consistency Checking:** Ensuring that the knowledge base doesn't contain contradictions.
    - **Property Inheritance:** Inferring properties of an object based on its category membership.
    - **Query Answering:** Answering questions about the relationships and properties of categories and objects.
- **Formalisms and Techniques:**
    - **Description Logics (DLs):** A family of formal knowledge representation languages that are particularly well-suited for representing and reasoning about categories and their relationships. They have well-defined semantics and decidable inference problems. DL reasoners (like FaCT++, Pellet, HermiT) are widely used for ontology reasoning.
    - **Semantic Networks:** Early graph-based representations where nodes represent concepts and links represent relationships, allowing for some forms of inheritance reasoning.
    - **Frame Systems:** Similar to semantic networks, but organize knowledge into "frames" with slots for attributes and relationships, often supporting inheritance and default values.
    - **First-Order Logic (FOL):** While very expressive, direct reasoning in FOL can be undecidable. DLs can be seen as decidable fragments of FOL.

Reasoning with Default Information

- **Definition:** This addresses the challenge of drawing conclusions in the presence of incomplete or uncertain information, where certain properties are typically true but can be overridden by exceptions.
- **The Problem:** Classical logic is monotonic – adding new information never invalidates previous conclusions. However, in real-world scenarios, we often make assumptions that might later be retracted.
    - **Example:** **"Birds typically fly." (Default)**
    - **"Penguins are birds."**
    - **"Penguins do not fly." (Exception)**
    - If we only know something is a bird, we might infer it flies. But if we then learn it's a penguin, we must retract that inference.

Definition of Classical Planning

Classical Planning is a core area in Artificial Intelligence (AI) that deals with finding a sequence of actions to achieve a desired goal in a structured environment.

Classical planning operates under a set of specific assumptions that simplify the problem, allowing for formal analysis and algorithmic solutions. **These assumptions are:**

- **Fully Observable:** The agent has complete and accurate knowledge of the current state of the world at all times. There's no uncertainty about the truth of propositions.

- **Deterministic Actions:** The outcome of every action is precisely known. Executing an action in a given state always leads to the same next state. There's no randomness.
- **Static World:** The world does not change on its own while the agent is deliberating or executing actions. Changes only occur as a result of the agent's actions.
- **Discrete States and Actions:** The world can be described by a finite set of discrete propositions (true/false facts), and actions have discrete, well-defined preconditions and effects.
- **Known Initial State:** The exact configuration of the world at the beginning of the planning problem is known.
- **Goal as a Set of Satisfied Conditions:** The goal is typically a conjunction of propositions that must be true in the final state. Any state satisfying these propositions is a goal state.

Classical planning problems are typically represented using formal languages like **STRIPS (STanford Research Institute Problem Solver)** or more commonly, **PDDL (Planning Domain Definition Language)**.

The task of a classical planner is to find a sequence of actions A1,A2,...,An such that when applied sequentially from the initial state, they transform the world into a state where all goal propositions are true

* **States:** A state is a set of propositions (facts) that are true in the current world.

* **Actions (Operators):** Each action is defined by:

  - **Preconditions:** A set of propositions that must be true for the action to be executed.
  - **Effects:** A set of propositions that become true (add effects) or false (delete effects) after the action is executed.

* **Initial State:** The starting set of true propositions.

* **Goal State:** A set of propositions that must be true in the final state.

Algorithms for Planning with State-Space Search

State-space search is a fundamental approach to **classical planning**. The planning problem is transformed into a search problem where:

- **Nodes:** Represent states of the world.
- **Edges:** Represent actions that transform one state into another.
- **Start Node:** The initial state.
- **Goal Nodes:** Any state that satisfies the goal conditions.

The algorithms used are often adaptations of general graph search algorithms:

- # Forward Search (Progression Search):
  - Starts from the initial state and applies applicable actions to explore new states until a goal state is reached.
  - **Algorithm Types:**
    - **Uninformed Search:**
      - **Breadth-First Search (BFS)**
      - **Depth-First Search (DFS)**
      - **Iterative Deepening Depth-First Search (IDS):** Combines the advantages of BFS (optimality) and DFS (memory efficiency) by performing a series of depth-limited DFS searches with increasing depth limits.
    - **Informed Search (Heuristic Search):**
      - **Heuristic Functions (h(s)):** Estimate the cost (number of actions) from the current state s to a goal state. A good heuristic can dramatically reduce the search space.
      - **Greedy Best-First Search:** Expands the node that appears closest to the goal based on the heuristic function h(s). Can be fast but not optimal.
      - **A* Search:** Combines the cost of reaching the current state g(s) with the heuristic estimate h(s) to evaluate nodes (f(s)=g(s)+h(s)). It is optimal and complete if the heuristic is admissible (never overestimates the true cost).

- # Backward Search (Regression Search):
  - **Concept:** Starts from the goal state and applies actions in reverse (regresses actions) to find a sequence that leads back to the initial state.
  - **Regression Operator:** For an action A and a state S′, the regression operator determines the set of states S such that applying A in S would result in S′.
  - **Advantages:** Can be more efficient if the goal is very specific and the initial state is vague, or if there are many possible initial states but a clear path to the goal. It avoids exploring actions that don't contribute to the goal.

Planning Graphs

Planning graphs are a specialized **data structure** used in **classical planning**, primarily to derive effective **heuristics for state-space search** and as a basis for some planners **(like GraphPlan).**

- **Structure:** A planning graph is a **layered, directed graph** with alternating **proposition layers** (Si) and **action layers** (Ai).
    - **S0 (Proposition Layer 0):** Represents the initial state, containing all propositions true in the initial state.
    - **Ai (Action Layer i):** Contains all ground actions whose preconditions are satisfied by the propositions in **Si**. It also includes "no-op" actions (persistence actions) for each proposition, indicating that it can remain true.
    - **Si+1 (Proposition Layer i+1):** Contains all propositions that could be made true by the effects of actions in Ai.
- **Edges:**
    - **Precondition Links:** From propositions in **Si** to actions in **Ai** if the proposition is a precondition of the action.
    - **Effect Links:** From actions in **Ai** to propositions in **Si+1** if the proposition is an effect of the action.
- **Mutex (Mutual Exclusion) Links:** A crucial aspect of planning graphs. They indicate propositions or actions that cannot be true or executed simultaneously at a given level.
    - **Action Mutexes:** Between two actions if:
        - **Inconsistent Effects:** An effect of one action negates an effect of the other.
        - **Interference:** An effect of one action negates a precondition of the other.
        - **Competing Needs:** A precondition of one action is mutex with a precondition of the other.
    - **Proposition Mutexes:** Between two propositions if every pair of actions that could achieve them are action mutexes at the preceding action layer.

Working Procedure :

1. **Expansion:** The graph is iteratively expanded level by level. It "levels off" when no new propositions or non-mutex relations appear in subsequent layers.

2. **Goal Check:** Once expanded, the graph can be checked for goal achievability: if all goal propositions are present in a proposition layer and are not mutually exclusive, a plan might exist.

3. **Heuristic Derivation:** Planning graphs are excellent for deriving admissible and informative heuristics. For example, the **level cost heuristic** estimates the cost to achieve a goal by finding the first level where all goal literals appear non-mutex.

4. **Plan Extraction (GraphPlan):** The GraphPlan algorithm uses the planning graph to extract a plan by performing a backward search *on the planning graph itself*. It tries to find a set of non-mutex actions at each level that can achieve the goals, working backward from the goal layer to the initial state

Other Classical Planning Approaches

- **Plan-Space Search (Partial-Order Planning - POP):**
    - Instead of searching through states, POP searches through a space of *partial plans*. A partial plan consists of actions, ordering constraints between them, and causal links
    - The planning process involves iteratively refining a partial plan by adding actions, ordering constraints, or causal links until a complete, valid plan is formed.
    - **Advantages:** Can be more flexible as it avoids committing to a total order of actions early on, potentially reducing backtracking.

- **Hierarchical Task Network (HTN) Planning:**
    - Instead of defining actions directly, HTN planning uses a hierarchy of tasks. High-level tasks are decomposed into subtasks or primitive actions until only executable primitive actions remain.
    - For each complex task, there are "methods" that specify how it can be decomposed into a network of subtasks.

    - **Advantages:** Excellent for problems with inherent hierarchical structure (e.g., "build a house" → "build walls" → "lay bricks"). It allows for incorporating domain knowledge and can guide the search more effectively.
- **Constraint Satisfaction Problem (CSP) / Satisfiability (SAT) Encoding:**
    - Transforms the planning problem into a CSP or a Boolean **Satisfiability (SAT)** problem.
    - **CSP:** States, actions, and their relationships are modeled as variables and constraints. A solution to the CSP directly corresponds to a plan.
    - **SAT:** Planning is encoded into a set of propositional logic clauses. A satisfying assignment to these clauses corresponds to a plan. Modern SAT solvers are highly optimized and can solve very large problems.
    - **Advantages:** Leverages powerful, generic off-the-shelf solvers. Can find optimal plans (shortest length) if encoded appropriately.

Analysis of Planning Approaches

- **State-Space Search (Forward/Backward):**
    - Conceptually straightforward, can leverage general search algorithms. Informed search with good heuristics is very powerful and forms the basis of many successful planners (e.g., FastForward, LAMA).
    - State explosion is a major issue without strong heuristics. Backward search can be tricky with complex goals.
    - The quality of the heuristic function is paramount for practical applicability.

- **Planning Graphs:**
    - Efficiently compute admissible heuristics. Can detect unsolvable problems early (when the goal becomes mutex). Forms the basis of efficient planners like Graph Plan, which guarantees shortest *length* plans (in terms of levels). Polynomial in size, unlike the exponential state space.
    - The plan extraction phase (backward search on the graph) can still be computationally expensive in the worst case. May not yield fully parallelizable plans if only focusing on level length.

- **Plan-Space Search (Partial-Order Planning):**
    - Generates partially ordered plans, which are more flexible for execution and potentially more robust to unexpected events. Avoids making premature commitments.
    - Can be harder to define and manage the search space. Reasoning about interactions between actions in a partial order can be complex. Less widely

used in modern domain-independent planners compared to state-space approaches.

- **HTN Planning:**
    - Integrates domain knowledge naturally, making it powerful for complex, structured problems. Can generate plans for highly hierarchical tasks.
    - Requires significant domain engineering (defining tasks and methods), making it less "general purpose" or "domain-independent."
- **CSP/SAT Encoding:**
    - Leverages highly optimized general-purpose solvers. Can find optimal solutions.
    - Encoding can be non-trivial. Can be difficult to interpret the "reasoning" behind a solution found by a black-box SAT solver. Scalability often depends heavily on the specific encoding and the performance of the solver.