CHAPTER 3

Unit III

Knowledge, Reasoning and Planning

Syllabus

3.1 Knowledge based agents

3.2 First order logic: syntax and Semantic, Knowledge Engineering in FOL Inference in FOL : Unification, Forward Chaining, Backward Chaining and Resolution

3.3 Planning Agent, Types of Planning: Partial Order, Hierarchical Order, Conditional Order

- Understanding theoretical or practical aspects of a subject is called as knowledge. We can gain knowledge through experience acquired based on the facts, information, etc. about the subject.
- After gaining knowledge about some subject we can apply that knowledge to derive conclusions about various problems related to that subject based on some reasoning.
- We have studied various types of agents in chapter 1. In this chapter we are going to see what is "knowledge based agent", with a very interesting game example.
- We are also going to study how do they store knowledge, how do they infer next level of knowledge from the existing set. In turn, we are studying various knowledge representation and inference methods in this chapter.

3.1 A Knowledge Based Agent

- As shown Fig. 3.1.1, a knowledge based agents can be described at different levels : **Knowledge Base (KB)** and an **Inference Engine.**

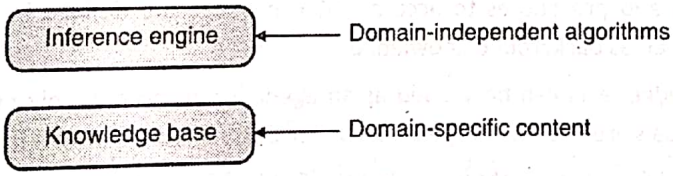


Fig. 3.1.1 : Levels of knowledge base

1. **Knowledge level:**
   - Knowledge level is a base level of an agent, which consists of domain-specific content.
   - In this level agent has facts/information about the surrounding environment in which they are working, it does not consider the actual implementation.
2. **Implementation level:**
   - Implementation level consists of domain independent algorithms. At this level, agents can recognize the data structures used in knowledge base and algorithms which use them. For example, propositional logic and resolution. (We will be learning about logic and resolution in this chapter)
   - Knowledge based agents are crucial to use in partially observable environments. Before choosing any action, knowledge based agents make use of the existing knowledge along with the current inputs from the environment in order to infer hidden aspects of the current state.

Scanned by CamScanner

- As we have learnt that knowledge base is a set of representations of facts/information about the surrounding environment (real world). Every single representation in the set is called as a **sentence** and sentences are expresses with the help of formal representation language. We can say that sentence is a statement which is a set of words that express some truth about the real world with the help of knowledge representation language.
- Declarative approach of building an agent makes use of TELL and ASK mechanism.
  - TELL the agent, about surrounding environment (what it needs to know in order to perform some action) TELL mechanism is similar to taking input for a system.
  - Then the agent can ASK itself what action should be carried out to get desired output. ASK mechanism is similar to producing output for a system. However, ASK mechanism makes use of the knowledge base to decide what it should do.
- TELL and ASK mechanism involve inference. When you run ASK function, the answer is generated with the help of knowledge base, based on the knowledge which was added with TELL function previously.
  - **TELL(K) :**Is a function that adds knowledge K to the knowledge base.
  - **ASK(K) :**Is a function that queries the agent about the truth of K.
- An agent carries out following operations: First, it TELLs the knowledge base about facts/information it perceives with the help of sensors. Then, it ASKs the knowledge base what action should be carried out based on the input it has received. Lastly, it performs the selected action with the help of effectors.

3.1.1 Architecture of a KB Agent

Knowledge based agents can be implemented at three levels namely, knowledge level, logical level and implementation level.

1.	Knowledge level	2.	Logical level
3.	Implementation level		

1. Knowledge level :

- It is the most abstract level of agent implementation. The knowledge level describes agent by saying what it knows. That is what knowledge the agent has as the initial knowledge.
- Basic data structures and procedures to access that knowledge are defined in his level. Initial knowledge of knowledge base is called as background knowledge.
- Agents at the knowledge level can be viewed as an agent for which one only need to specify what the agent knows and what its goals are in order to specify its behaviour, regardless of how it is to be implemented.
- **For example :**A taxi driving agent might know that the Golden Gate Bridge connects San Francisco with the Marin county.

2. Logical level :

- At the logical level, the knowledge is encoded into sentences. This level uses some formal language to represent the knowledge the agent has. The two types of representations we have are propositional logic and first order or predicate logic.
- Both these representation techniques are discussed in detail in the further sections.
- **For example:** Links(Golden Gate Bridge, San Francisco, Marin County)

3. Implementation level :

- In implementation level, the physical representation of logical level sentences is done. This level also describes data structures used in knowledge base and algorithms that used for data manipulation.

Scanned by CamScanner

- **For example :** Links(Golden Gate Bridge, San Francisco, Marin County)

```
function KB – Agent (percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially o, indicating time
  TELL (KB, MAKE – PERCEPT-SENTENCE(percept, t))
  action← ASK (KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action,t))
  t←t + 1
  returns action
```

Fig. 3.1.2 : General function of knowledge based agent

- Fig. 3.1.2 is the general implementation of knowledge based agent. TELL and ASK are the sub procedures implemented to perform the respective actions.
- **The knowledge base agent must be able to perform following tasks :**
  - Represent states, actions, etc.
  - Incorporate new precepts.
  - Update internal representations of the world.
  - Deduce hidden properties of the world.
  - Deduce appropriate actions.

3.2 The WUMPUS World Environment

- You have learnt about vacuum world problem, block world problem so far. Similarly we have WUMPUS world problem. Fig.3.2.1 shows the WUMPUS world.
- WUMPUS is an early computer game also known as "Hunt the Wumpus". WUMPUS was developed by Gregory Yob in 1972/1973. It was originally written in **BASIC (Beginner's All-purpose Symbolic Instruction Code).**
- WUMPUS is a map-based game. Let's understand the game :
  - WUMPUS world is like a cave, which represents number of rooms, rooms, which are connected by passage ways. We will take a 4×4 grid to understand the game.
  - WUMPUS is a monster who lives in one of the rooms of the cave. WUMPUS eats the player (agent) if player (agent) comes in the same room. Fig. 3.2.1 shows that room (3, 1) where WUMPUS is staying.
  - Player (agent) starts from any random position in cave and has to explore the cave. We are starting from (1, 1) position.
- There are various sprites in the game like pit, stench, breeze, gold, and arrow. Every sprite has some feature. Let's understand this one-by-one :
  - Few rooms have bottomless pits, which trap the player (agent) if he comes to that room. You can see in the Fig. 3.2.1 that room (1,3), (3,3) and (4,4) have bottomless pit. Note that even WUMPUS can fall into a pit.
  - Stench experienced in a room, which has a WUMPUS in its neighbourhood room. See the Fig. 3.2.1, here room (2,1), (3,2) and (4,1) have Stench.
  - Breeze is experienced in a room, which has a pit in its neighbourhood room. Fig. 3.2.1 shows that room (1,2), (1,4), (2,3), (3,2), (3,4) and (4,3) consists of Breeze.
  - Player (Agent) has arrows and he can shoot these arrows in straight line to kill WUMPUS.

Scanned by CamScanner

- One of the rooms consists of gold, this room glitters. Fig.3.2.1 shows that room (3,2) has Gold.

– Apart from above features player (agent) can accept two types of percepts which are: Bump and scream. A bump is generated if player (agent) walks into a wall. While a sad scream created everywhere in the cave when the WUMPUS is killed.

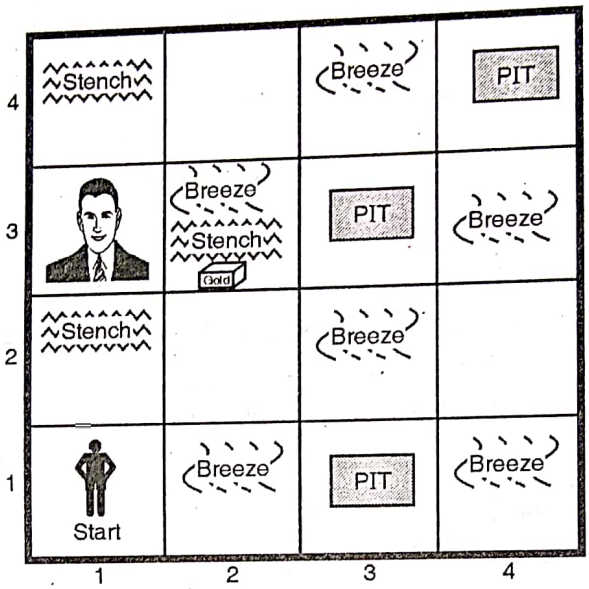


Fig. 3.2.1 : The WUMPUS World

3.2.1 Description of the WUMPUS World

– An agent receives percepts while exploring the rooms of cave. Every percepts can be represented with the help of five element list, which is [stench, breeze, glitter, bump, scream]. Here player (agent) cannot perceive its own location.

– If the player (agent) gets percept as [Stench, Breeze, None, None, None]. Then it means that there is a stench and a breeze, but no glitter, no bump, and no scream in the WUMPUS world at that position in the game.

– Let's take a look at the actions which can be performed by the player(agent) in WUMPUS World :

- **Move :** To move in forward direction,
- **Turn :** To turn right by 90 degrees or left by 90 degrees,
- **Grab :** To pick up gold if it is in the same room as the player(agent),
- **Shoot :** To Shoot an arrow in a straight line in the direction faced by the player (agent).

– These actions are repeated till the player (agent) kills the WUMPUS or if the player (agent) is killed. If the WUMPUS is killed then it is a winning condition, else if the player(agent) is killed then it is a losing condition and the game is over.

– Game developer can keep a restriction on the number of arrows which can be used by the player(agent). So if we allow agent to have only one arrow, then only the first shoot action will have some effect. If this shoot action kills the WUMPUS then you win the game, otherwise it reduces the probability of winning the game.

– Lastly there is a die action : It takes places automatically if the agent enters in a room with a bottomless pit or in a room with WUMPUS. Die action is irreversible.

Goal of the game

– Main aim of the game is that player (agent) should grab the gold and return to starting room (here its (1,1)) without being killed by the monster (WUMPUS).

Scanned by CamScanner

- Award and punishment *points* are assigned to a player (Agent) based on the actions it performs. **Points can be given as follows:**
  - 100 points are awarded if player (agent) comes out of the cave with the gold.
  - 1 point is taken away for every action taken.
  - 10 points are taken away if the arrow is used.
  - 200 points are taken away if the player (agent) gets killed.

3.2.2 PEAS Properties of WUMPUS World

MU - May 13, Dec. 14

Q. Give PEAS descriptors for WUMPUS world **(May 13, Dec. 14, 3 Marks)**

1. **Performance measure**
   - +100 for grabbing the gold and coming back to the starting position,
   - – 200 if the player(agent) is killed.
   - – 1 per action,
   - – 10 for using the arrow.
2. **Environment**
   - Empty Rooms.
   - Room with WUMPUS.
   - Rooms neighbouring to WUMPUS which are smelly.
   - Rooms with bottomless pits
   - Rooms neighbouring to bottomless pits which are breezy.
   - Room with gold which is glittery.
   - Arrow to shoot the WUMPUS.
3. **Sensors (assuming a robotic agent)**
   - Camera to get the view
   - Odour sensor to smell the stench
   - Audio sensor to listen to the scream and bump
4. **Effectors (assuming a robotic agent)**
   - Motor to move left, right
   - Robot arm to grab the gold
   - Robot mechanism to shoot the arrow

The WUMPUS world agent has following characteristics :

1. Fully observable
2. Deterministic
3. Episodic
4. Static
5. Discrete
6. Single agent

Scanned by CamScanner

3.2.3 Exploring a WUMPUS World

– Let's try to understand the WUMPUS world problem in step by step manner. Keep Fig. 3.2.2 as a reference figure.

4.1	4.2	4.3	4.4
3.1	3.2	3.3	3.4
2.1	2.2	2.3	2.4
1.1 [A] OK	1.2 OK	1.3	1.4

[A] - Agent
B – Breeze
G - Glitter, Gold
OK – Safe square
P - Pit
S – Stench
V – Visited
W –Wumpus

Fig. 3.2.2(a) : WUMPUS world with player in room (1,1)

– The knowledge base initially contains only the rules (facts) of the WUMPUS world environment.

Step 1 : Initially the player(agent) is in the room (1,1). See Fig. 3.2.2(a).

The first percept received by the player is [none, none, none, none, none]. (remember percept consists of [stench, breeze, glitter, bump, scream])

Player can move to room(1,2) or (2,1) as they are safe cells.

Step 2 : Let us move to room (1,2). See Fig.3.2.2(b).

4.1	4.2	4.3	4.4
3.1	3.2	3.3	3.4
2.1	2.2	2.3	2.4
1.1 V OK	1.2 [A] B OK	1.3 P?	1.4

[A] - Agent
B – Breeze
G - Glitter, Gold
OK – Safe square
P - Pit
S – Stench
V – Visited
W –Wumpus

Fig. 3.2.2(b) : WUMPUS world with player in room (1,2)

– As room (1,1) is visited you can see "V" mark in that room. The player receives following percept :[none, breeze, none, none, none].

– As breeze percept is received room (1,2) is marked with "B" and it can be predicted that there is a bottomless pit in the neighboring room.

– You can see that room (1,3) and room (2,2) is marked with "P?". So room (1,3) and (2,2) is not safe to move in. Thus player should return to room (1,1)and try to find other, safe room to move to.

Scanned by CamScanner

Step 3 :

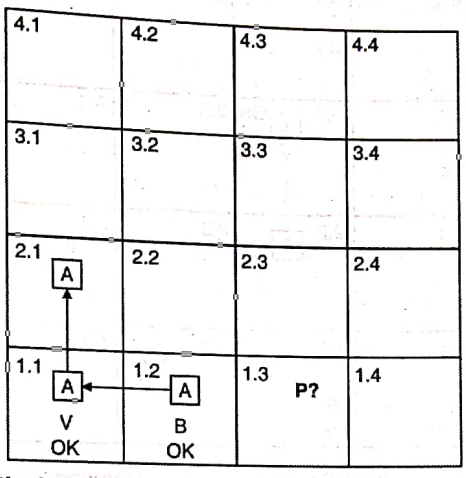


Fig. 3.2.2(c) WUMPUS world with player moving back to room (1,1) and then moves to other safe room (2,1).

- As seen in Fig.3.2.2(c). Player in now in room (2,1), where it receives a percept as follows : [stench, none, none, none, none] which means that there is a WUMPUS in neighboring room (i.e. either room (2,2) or (3,1) has WUMPUS).
- As we did not get breeze percept in this room, we can understand that room (2,2) cannot have any pit and from step 2 we can understand that room (2,2) cannot have WUMPUS because room (1,2) did not show stench percept.
- Thus room(2,2) is safe to move in.

Step 4 : Player receives [none, none, none, none, none] percept when it comes to room (2,2). From Fig. 3.2.2(d) you can understand that room (2,3) and room (3,2) are safe to move in.

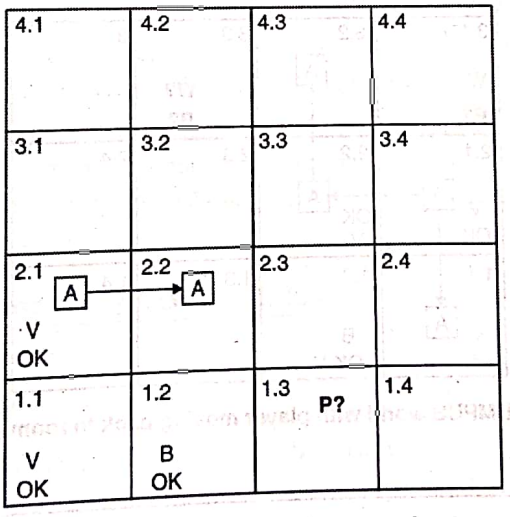


Fig. 3.2.2(d) : WUMPUS world with player moving to room (2,2)

Step 5: Let's move to room (3,2). Here, player receives [stench, breeze, glitter, none, none] percept. See Fig. 3.2.2(e). Field 1 of the percept shows that room (3,1), (3,3) and (4,2) can have WUMPUS. Field 2 of the percept shows that room (3,1), (3,3), (2,2) and (4,2) can have bottomless pit. Field 3 of the percept shows that room (3,2) has gold. So, the player grabs the gold first. As the aim of this game is to grab the gold and go back to the starting position, without being killed by the WUMPUS.

Scanned by CamScanner

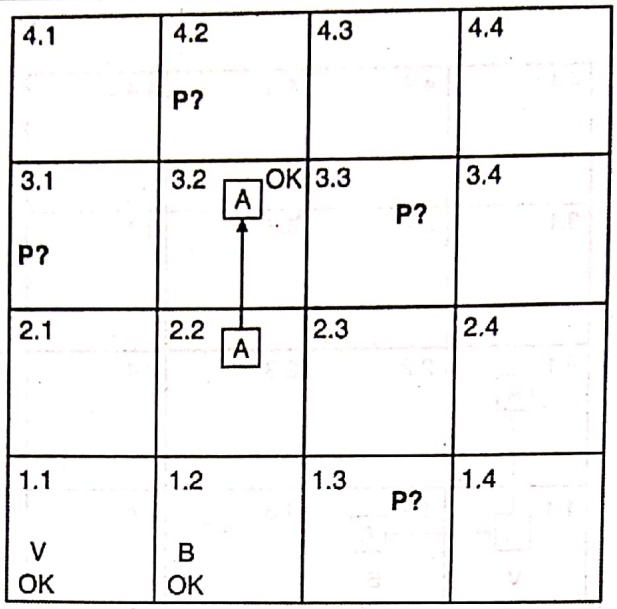


Fig. 3.2.2(e) : WUMPUS world with player moving to room (3,2)

Now, we have to go back to the starting position i.e. room (1,1) without getting killed by WUMPUS. From steps 1, 2, 3 and 4 We know that room (1,1), (1,2), (2,1) and (2,2) are safe rooms. so, we can go back to room (1,1) by following any of the two paths: i.e. (2,2), (2,1), (1,1) or (2,2), (1,2), (1,1).

Step 6: As can be seen in Fig. 3.2.2(f). We will go from room (2,2) to room (2,1) and from room (2,1) to room (1,1). Thus we won the WUMPUS World game!!!

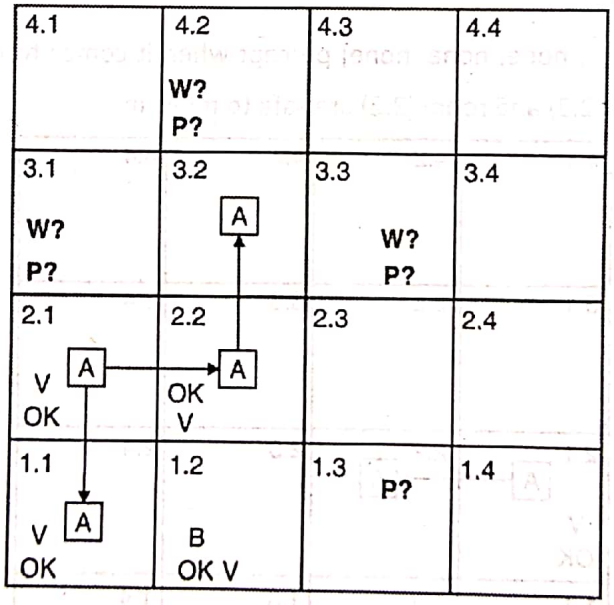


Fig.3.2.2(f) : WUMPUS world with player moving back to room (1,1) with gold

3.3 Logic

- Logic can be called as **reasoning** which is carried out or it is a review based on strict rule of validity to perform a specified task.
- In case of intelligent systems we say that any of logic's particular form cannot bind logical representation and reasoning, they are independent of any particular form of logic.
- Make a note that logic is beneficial only if the knowledge is represented in small extent and when knowledge is represented in large quantity the logic is not considered valuable.
- Fig. 3.3.1 depicts that sentences are physical configurations of an agent, also it is shows that sentences need sentence. This means that reasoning is a process of forming new physical configurations from old ones.

Scanned by CamScanner

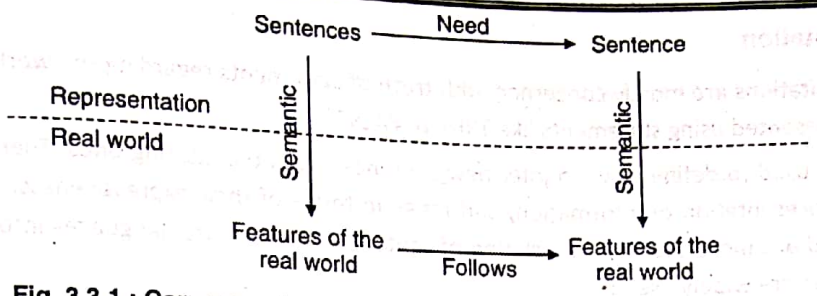


Fig. 3.3.1 : Correspondence between real world and its representation

- Logical reasoning should make sure that the new configurations represent features of the world that actually follow the features of the world that the old configurations represented

3.3.1 Role of Reasoning in AI

- Fig.3.3.2 shows how logic can be seen as a knowledge representation language. There are various levels to the logic and most fundamental type of logic is propositional logic.

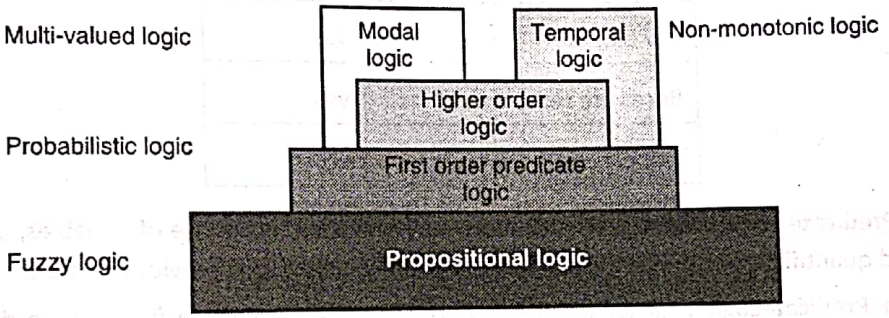


Fig. 3.3.2 : Logic as Knowledge Representation language

- Propositional logic can be considered at fuzzy logic level, where rules are values between range of 0 and 1. Next level is also called as probabilistic logic level using which first order predicate logic is implemented.
- In this Fig. 3.3.2, there are two more levels above higher order logic which are multi-valued and non-monotonic logic levels and they consist of modal logic and temporal logic respectively. All these types of logic are basic building blocks of intelligent systems and they all use reasoning in order to represent sentences. Hence reasoning plays a very important role in AI.

3.4 Representation of Knowledge using Rules

MU – May 13, Dec. 14, May 15

Q. Explain various methods of knowledge representation with example. (May 13, Dec. 14, May 15, 10 Marks)

- Knowledge can be considered to be represented at generally two levels :
  (i) **Knowledge level :**This level describes the facts.
  (ii) **Symbol level :**This level deals with using the symbols for representing the objects, which can be manipulated in programs.
- **Knowledge can be represented using the following rules :**
  (a) Logical representations
  (b) Production rule representations
  (c) Semantic networks
  (d) Frame representations

Scanned by CamScanner

(a) Logical representation

- The logical representations are mostly concerned with truth of statements regarding the world. These statements are most generally represented using statements like TRUE or FALSE.
- Logic is successfully used to define ways to infer new sentences from the existing ones. There are certain logics that are used for the representation of information, and range in terms of their expressiveness. There are logic that are more expressive and are more useful in translation of sentences from natural languages into the logical ones. There are several logics that are widely used :-

1. **Propositional logic :** These are restricted kinds that make use of propositions (sentences that are either true or false but not both) which can be either true or false. Proposition logic is also known as propositional calculus, sentential calculus or boolean algebra.

   All propositions are either true or false, **For example :** .

   (i) Leaves are green (ii) Violets are blue.

Sentence	Truth Value	Proposition
Sky is blue	true	yes
Roses are red	true	yes
2+2=5	false	yes

2. **First Order Predicate Logic :** These are much more expressive and make use of variables, constants, predicates, functions and quantifiers along with the connectives explained already in previous section.
3. **Higher Order PredicateLogic :** Higher order predicate logic is distinguished from first order predicate logic by using additional quantifiers and stronger semantics.
4. **Fuzzy Logic :** These indicate the existence of in between TRUE and FALSE or fuzziness in all logics.
5. **Other Logic :** These include multiple valued logic, modal logics and temporal logics.

(b) Production rule representation

One of the widest used methods to represent knowledge is to use production rules, it is also known as IF-THEN rules.

Syntax :

```
IF condition THEN action
IF premise  THEN conclusion
IF proposition p1 and proposition p2 are TRUE
THEN proposition p3 is TRUE
```

Example :

IF pressure is high, THEN volume is small.

IF the road is slippery, THEN driving is dangerous.

- Some of the benefits of IF-THEN rules are that they are modular, each defining a relatively small and, at least in principle, independent piece of knowledge. New rules may be added and old ones deleted usually independently of other rules.
- Production rules are simple but powerful forms of representing knowledge, they provide flexibility for combining procedural and declarative representations in a unified manner. The major advantage of production rules are that they are modular, independent of other rules with the provision for addition new rules and deleting older ones.

Scanned by CamScanner

(c) Semantic networks

- These represent knowledge in the form of graphical networks, since graphs are easy to be stored inside programs as they are concisely represented by nodes and edges.
- A semantic network basically comprises of *nodes* that are named and represent concepts, and labelled *links* representing relations between concepts. Nodes represent both types and tokens.
- For example, the semantic network in Fig. 3.4.1 expresses the knowledge to represent the following data :
  - Tom is a cat.
  - Tom caught a fish.
  - Tom is grey in color.
  - Tom is owned by Sam.
  - Tom is a Mammal.
  - Fish is an Animal.
  - Cats love Milk.
  - All mammals are animals.

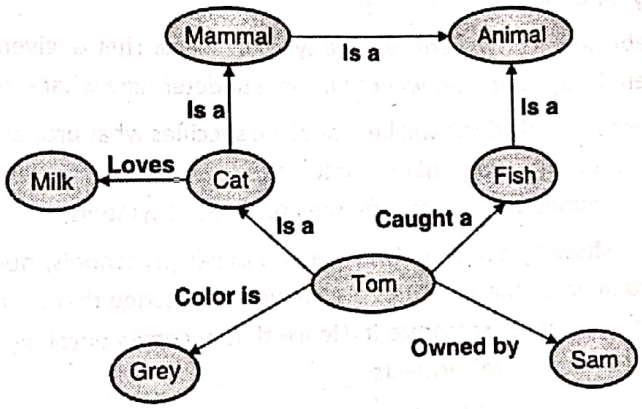


Fig. 3.4.1

- **Conceptual Graph :** It is a recent scheme used for semantic network, introduced by John Sowa, has a finite, connected, bipartite graph. The nodes represent either concepts or conceptual relations. It differs from the previous method that it does not use labelled arcs. **For example:** Ram, Laxman and Bharat are Brothers or cat color is grey can be represented as shown.

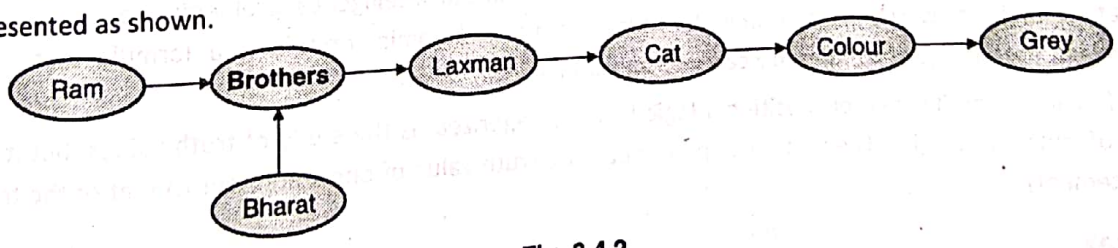


Fig. 3.4.2

(d) Frame representation

- This concept was introduced by Marvin Minsky in 1975. They are mostly used when the task becomes quite complex and needs more structured representation. More structured the system becomes more would be the requirement of using frames which would prove beneficial. Generally frames are record like structures that consists of a collection of slots or attributes and the corresponding slot values.
- Slots can be of any size and type. The slots have names and values (subfields) called as facets. Facets can have names or numbers too. A simple frame is shown in the Fig. 3.4.2 for a person Ram,

Scanned by CamScanner

- (Ram)
- (PROFESSION (VALUE professor)
- (AGE(VALUE 50))
- (WIFE(VALUE sita))
- (CHILDREN(VALUE luv kush))
- (ADDRESS (STREET(VALUE 4C gb road)))
- CITY(VALUE banaras))
- (STATE(VALUE mh))
- (ZIP(VALUE400615))

3.4.1 Ontology

- Ontology is study about what kind of things or entities exist in the universe. In AI, ontology is the specification of conceptualizations, used to help programs and humans to share knowledge about a particular domain. In turn, ontology is a set of concepts, like entity, relationships among entities, events that are expressed in a uniform way in order to create a vocabulary for information exchange.
- An ontology should also enable a person to verify what a symbol means. That is, given a concept, they want to be able to find the symbol, and, given the symbol, they want to be able to determine what it means.
- Typically, it specifies what types of individuals will be modelled, specifies what properties will be used, and gives some axioms that restrict the use of that vocabulary. Ontologies are usually written independently of a particular application and often involve a community to agree on the meanings of symbols
- For example: Consider a map showing hotels, railway station, buildings, schools, hospitals in a particular locality. In this map the symbols used to indicate these entities are enough to describe them. Hence the community who knows the meaning of these symbols can easily recognize it. Hence that becomes ontology of that map. In this ontology, it may define a building as human-constructed artifacts.
- It may give some restriction on the size of buildings so that shoeboxes cannot be buildings or that cities cannot be buildings. It may also state that a building cannot be at two geographically dispersed locations at the same time.

3.5 Propositional Logic (PL)

- **Propositional Logic(PL)** is simple but powerful for some artificial intelligence problems. You have learnt simple mathematical logic in which uses atomic formulas are used. (Atomic formula is a formula that has no strict sub-formulas). Atomic logic formulas are called propositions.
- In case of artificial intelligence propositional logic is not categorized as the study of truth values, but it is based on relativity of truth values. (i.e. The relationship between the truth value of one statement to that of the truth value of other statement)

3.5.1 Syntax

Basic syntax followed by the propositional logic can be given as follows:

- Propositional symbols are denoted with capital letters like: A, B, C, etc.
- Propositional logic constants have a truth value generally truth values have a crisp nature (i.e. 0 (false) and 1 (true)). But for fuzzy logic truth values can vary in the range of 0 and 1.
- Propositional logic make use of wrapping parenthesis while writing atomic sentence. It is denoted as '(....)'.
- Literal is an atomic sentence or it can be negation of atomic sentence. (A, ¬A)

Scanned by CamScanner

If A is a sentence, then ¬A is a sentence.

- Propositional logic makes use of relationships between propositions and it is denoted by connectives, if A and B are propositions. Connectives used in proposition logic can be seen in the Table 3.5.1.

Table 3.5.1 : Connectives used in Propositional logic

Connective symbol	Name of the Connective symbol	Relationship between Propositional symbols	Name of the Relationship between Propositional symbols
∧	And	A ∧ B	Conjunction
∨	Or	A ∨ B	Disjunction
¬	Not	¬A	Negation
⇒	Implies	A ⇒ B	Implication/ conditional
⇔	is equivalent/ if and only if	A ⇔ B	Biconditional

- To define logical connectives truth tables are used. Truth table 3.5.2 shows five logical connectives.

Table 3.5.2

A	B	A ∧ B	A ∨ B	¬A	A ⇒ B	A ⇔ B
False	false	false	False	true	true	true
False	true	false	True	true	true	false
True	false	false	True	false	false	false
True	true	true	True	false	true	true

- Take an example, where **A ∧ B**, i.e. Find the value of **A ∧ B** where A is true and B is false. Third row of the Table 3.5.2 shows this condition, now see third row of the third column where, **A ∧ B** shows result as false. Similarly other logical connectives can be mapped in the truth table.

3.5.2 Semantics

- World is set of facts which we want to represent to form propositional logic. In order to represent these facts propositional symbols can be used where each propositional symbol's interpretation can be mapped to the real world feature.
- Semantics of a sentence is meaning of a sentence. Semantics determine the interpretation of a sentence. **For example :**You can define semantics of each propositional symbol in following manner:

1. A means "It is hot"
2. B means "It is humid", etc.

- Sentence is considered true when its interpretation in the real world is true. Every sentence results from a finite number of usages of the rules. For example, if A and B are sentences then (A ∧ B), (A ∨ B), (B → A) and (A ↔ B) are sentences. The knowledge base is a set of sentences as we have seen in previous section.
- Thus we can say that real world is a model of the knowledge base when the knowledge base is true for that world. In other words a model can be thought of as a truth assignment to the symbols.

Scanned by CamScanner

- If truth values of all symbols in a sentence are given then it can be evaluated for determining its truth value (i.e. we can say if it is true or false).

3.5.3 What is Propositional Logic ?

- A ∧B and B∧A should have same meaning but in natural language words and sentences may have different meanings. Say for an example,
  1. Radha started feeling feverish and Radha went to the doctor.
  2. Radha went to the doctor and Radha started feeling feverish.
- Here, sentence 1 and sentence 2 have different meanings.
- In artificial intelligence propositional logic is a relationship between the truth value of one statement to that of the truth value of other statement.

3.5.4 PL Sentence - Example

Take example of a weather problem.

- Semantics of each propositional symbol can be defined as follows:
  - Symbol A is a sentence, which means "It is hot".
  - Symbol B is a sentence, which means "It is humid".
  - Symbol C is a sentence, which means "It is raining".
- We can also choose symbols which are easy to understand, like:
  - HT for "It is hot".
  - HM for"It is humid".
  - RN for"It is raining".
- If you have HM→HT, then that means "If it is humid, then it is hot".
- If you have(HT ∧ HM) → RN then it means "If it is hot and humid, then it is raining" and so on.
- First we have to create the possible mode ls for a knowledge base. To do this we need to consider all the possible assignments of true or false values for Sentence A, B and C. Then verify the truth table for the validity. There can be total 8 possibilities as shown below:

Sentence HT	Sentence HM	Sentence RN	Validity
False	False	False	Valid
False	False	True	Valid
False	True	False	Not Valid
False	True	True	Not Valid
True	False	False	Valid
True	False	True	Valid
True	True	False	Not Valid
True	True	True	Valid

- Now, if the knowledge base is [HM, HM→HT, (HT∧HM) → RN](i.e. ["It is humid", "If it is humid, then it is hot" , "If it is hot and humid, then it is raining"]), then "True -True - True" is the only possible valid model.

Scanned by CamScanner

Tautology and Contradiction

- **Tautology** means valid sentence. It is a sentence which is true for all the interpretations. **For example :** $(A \vee \neg A)$ ("*A* or not *A*") : "It is hot or It is not hot"
- **Contradiction** means an inconsistent sentence. It is a sentence which is false for all the interpretations. **For example :** $A \wedge \neg A$ ("*A* and not *A*") : "It is hot and it is not hot."
- **X entails Y,** is shown as X |= Y. It means that whenever sentence X is True, sentence Y will be True. **For Example :** if, X= Priya is Pooja's Mother's Sister and Y = Priya is Pooja's Aunty. Then X |= Y (X entails Y).

3.5.5 Inference Rules

- New sentences are formed with the logical inference. **For example :** If A = B and B = C then A = C. You must have come across this example many times it implies that if knowledge base has "A = B" and "B = C" then we can infer that "A = C".
- In short inference rule says that new sentence can be create by logically following the set of sentences of the knowledge base.

Table 3.5.3 : Inference Rules

Inference Rules	Premise (KB)	Conclusion
Modus Ponens	X, X→Y	Y
Substitution	X →Z &Y → Z	X = Y
Chain rule	X → Y, Y→Z	X →Z
AND introduction	X, Y	X ^ Y
Transposition	X →Y	~X →~Y

- Entailment is represented as : KB |= Q and Derivation is represented as : KB |- Q.
- There are two types of inference rules :

1. Sound inference
2. Complete inference

1. Sound Inference

- Soundness property of inference says that, if "X is derived from the knowledge base" using given set of protocols of inference, then "X is entailed by knowledge base". Soundness property can be represented as : "If KB |- X then KB |= X".
- For **Modus Ponens (MP)** rule we assume that knowledge base has [A, A → B], from this we can conclude that knowledge base can have B. See following truth table :

A	B	A → B	Valid?
TRUE	TRUE	TRUE	Yes
TRUE	FALSE	FALSE	Yes
FALSE	TRUE	TRUE	Yes
FALSE	FALSE	TRUE	Yes

Scanned by CamScanner

In general,

For atomic sentences p_i, p_i', and q, where there is a substitution Θ such that

$$\text{SUBST}(\Theta, p_i) = \text{SUBST}(\Theta, p_i') \quad \text{for all i,}$$

$$\frac{p_1', p_2', p_3', \ldots p_n', (p_1 \wedge p_2 \wedge p_3 \wedge \ldots \ldots .. p_n \Rightarrow q)}{\text{SUBST}(\Theta, q)}$$

N + 1 premises = N atomic sentences + one implication.

Example :

A : It is rainy.

B : I will stay at home.

A→B : If it is rainy, I will stay at home.

Modus Tollens

When B is known to be false, and if there is a rule "if A, then B," it is valid to conclude that A is also false.

2. Complete inference

- Complete inference is converse of soundness. Completeness property of inference says that, if "X is entailed by knowledge base" then "X can be derived from the knowledge base" using the inference protocols.
- Completeness property can be represented as: " If KB |= Q then KB |- Q".

3.5.6 Horn Clause

- Clauses are generally written as sets of literals. Horn clause is also called as **horn sentence**. In a horn clause a conjunction of 0 or more symbols is to the left of "→" and 0 or 1 symbols to the right. See following formula :

  $A_1 \wedge A_2 \wedge A_3 \ldots \wedge A_n \rightarrow B_m$ where n>=0 and m is in range{0,1}

- There can be following special cases in horn clause in the above mentioned formula :
  - For n=0 and m=1: A (This condition shows that assert A is true)
  - For n>0 and m=0: $A \wedge B \rightarrow$ (This constraint shows that both A and B cannot be true)
  - For n=0 and m=0: (This condition shows empty clause)
- Conjunctive normal form is a conjunction of clauses and by its set of clauses it is determined up to equivalence. For a horn clause conjunctive normal form can be used where, each sentence is a disjunction of literals with at most one non-negative literal as shown in the following formula : $\neg A_1 \vee \neg A_2 \vee \neg A_3 \ldots \vee \neg A_n \vee B$
- This can also be represented as : $(A \rightarrow B) = (\neg A \vee B)$

Significance of horn logic

- Horn sentences can be used in first order logic. Reasoning processes is simpler with horn clauses. Satisfiability of a propositional knowledge base is NP complete. (Satisfiability means the process of finding values for symbols which will make it true).
- For restricting knowledge base to horn sentences, satisfiability is in A. Due to this reason, first order logic horn sentences are the basis for prolog and data log languages.
- Let's take one example which gives entailment for horn formulas.
- Find out if following horn formula is satisfiable?

  $(true \rightarrow X) \wedge (X \wedge Y \rightarrow Z) \wedge (Z \mid W) \wedge (Z \wedge W \rightarrow false) \wedge (true \rightarrow Y)$

Scanned by CamScanner

- From the above equation, we entail if the query atom is false. Equation shows that there are clauses which state that true →X and true →Y, so we can assign X and Y to true value (i.e. true →X ∧Y).
- Then we can say that all premises of X∧Y→Z are true, based on this information we can assign Z to true. After that we can see all premises of Z→W are true, so we can assign W to true.
- As now all premises of Z∧W →false are true, from this we can entail that the query atom is false. Therefore, the horn formula is not satisfiable.

3.5.7 Propositional Theorem Proving

- Sequence of sentences form a "Proof". A sentence can be premise or it can be a sentence derived from earlier sentences in the proof based on the inference rule. Whatever we want to prove is called as a query or a goal. Query/goal is the last sentence of the theorem in the proof.
- Take Example of the "weather problem" which we have seen above.
  - HT for "It is hot".
  - HM for "It is humid".
  - RN for "It is raining".

1.	HM	Premise (initial sentence)	"It's humid"
2.	HM →HT	Premise(initial sentence)	"If it's humid, it's hot"
3.	HT	Modus ponens(1,2) (sentence derived from 1 and 2)	"It's hot"
4.	(HT ∧HM)→RN	Premise(initial sentence)	"If it's hot and humid, it's raining"
5.	HT ∧HM	And introduction(1,3)	"It's hot and humid"
6.	RN	Modusponens(4,5)(sentence derived from 4 and 5)	"It's raining"

3.5.8 Advantages of Propositional Logic

- Propositional logic is a simple knowledge representation language.
- It is sufficient and efficient technique for solving some artificial intelligence based problems.
- Propositional logic forms the foundation for higher logics like **First Order Logic (FOL)**, etc.
- Propositional logic is NP complete and reasoning is decidable.
- The process of inference can be illustrated by PL.

3.5.9 Disadvantages of Propositional Logic

- Propositional logic is cannot express complex artificial intelligence problems.
- Propositional logic can be impractical for even small worlds, think about WUMPUS hunter problem.
- Even if we try to make use of propositional logic to express complex artificial intelligence problems, it can be very wordy and lengthy.
- PL is a weak knowledge representation language because :
  - With PL it is hard to identify if the used entity is "individual". For example : If there are entities like : Priya, Mumbai, 123, etc.

Scanned by CamScanner

- PL cannot directly represent properties of individual entities or relations between individual entities. For example, Pooja is tall.
- PL cannot express specialization, generalizations, or patterns, etc. **For example:** All rectangles have 4 sides.

3.6 First Order Predicate Logic

MU - May 14

Q. Write a short note on predicate logic. (May 14, 5 Marks)

- Because of the inadequacy of PL discussed above there was a need for more expressive type of logic. Thus **First-Order Logic (FOL)** was developed. FOL is more expressive than PL, it can represent information using relations, variables and quantifiers, e.g., which was not possible with propositional logic.
  - "Gorilla is Black" can be represented as :

    Gorilla(x) →Black(x)
  - "It is Sunday today" can be represented as :

    today(Sunday)
- First Order Logic(FOL)is also called as **First Order Predicate Logic (FOPL)**. Since FOPL is much more expressive as a knowledge representation language than PL it is more commonly used in artificial intelligence.

3.6.1 Syntactic Elements, Semantic and Syntax

- FOL symbol can be a constant term, a variable term or a function.
- Assuming that "X" is a domain of values, we can define a term with following rules :
  1. **Constant term:** It is a term with fixed value which belongs to the domain.
  2. **Variable term:** It is a term, which can be assigned values in the domain.
  3. **Function :** Say "f" is a function of "n" arguments. If we assume that $t_1, t_2, .., t_n$ are terms then $f(t_1, t_2, .., t_n)$ is also called as a term.
- All the terms are generated by applying the above three protocols.
- First order predicate logic makes use of propositional logic as a base logic, so the connectives used in PL and FOPL are common. Hence, it also supports ∧ conjunction, ∨ disjunction, negation, ⇒ implication and ⇔ double implication.
- **Ground Term:** If a term does not have any variables it is called as a **ground term**. A sentence in which all the variables are quantified is called as a "well-formed formula".
  - Every ground term is mapped with an object.
  - Every condition (predicate) is mapped to a relation.
  - A ground atom is considered as true if the predicate's relation holds between the terms' objects.
  - **Rules in FOL:**In predicate logic rule has two parts predecessor and successor. If the predecessor is evaluated to TRUE successor will be true. It uses the implication → symbol. Rule represents If-then types of sentences.
- Example: The sentence "If the bag is of blue colour, I will buy it." Will be represented as colour (bag, blue)→buy(bag).

Quantifiers

Apart from these connectives FOPL makes use of quantifiers. As the name suggests they quantify the number of variables taking part in the relation or obeying the rule.

Scanned by CamScanner

1. **Universal Quantifier '∀'**
   - Pronounced as "for all" and it is applicable to all the variables in the predicate
   - "∀x A" means A is true for every replacement of x.
   - Example: "Every Gorilla is Black" can be represented as :
     "∀x (Gorilla(x) →Black(x))

2. **Existential Quantifier '∃'**
   - Pronounced as "there exists"
   - "∃x A" means A is true for at least one replacement of x.
   - Example: "There is a white dog" can be represented as,
     ∃ x (Dog(X) ^ white(X))

Note :

1. Typically, ⇒ is the main connective with∀

   Example:"Everyone at MU is smart" is represented as

   ∀x At(x, MU) → smart(x)

2. Typically, ∧ is the main connective with∃

   Example: Someone killed the cat and is guilty.

   ∃x killed(x, cat) ∧ guilty(x)

   - **Equality :** $term_1 = term_2$is true under a given interpretation if and only if $term_1$and $term_2$refer to the same object
   - **Example :** Richard has at least two brothers

   ∃x,∃y Brother(x, Richard) ∧ Brother(y, Richard) ∧¬ (x = y)

3.7 Comparison between Propositional Logic and First Order Logic

Sr. No.	Propositional logic (PL)	Predicate logic (FOL)
1.	PL cannot represent small worlds like vacuum cleaner world.	FOL can very well represent small worlds' problems.
2.	PL is a weak knowledge representation language	FOL is a strong way of representing language.
3.	Propositional Language uses propositions in which the complete sentence is denoted by a symbol.	FOL uses predicated which involve constants, variables, functions, relations.
4.	PL cannot directly represent properties of individual entities or relations between individual entities. e.g. Meera is short.	FOL can directly represent properties of individual entities or relations between individual entities using individual predicates using functions. E.g. Short(Meera)
5.	PL cannot express specialization,generalizations, or patterns, etc. e.g. All rectangles have 4 sides.	FOL can express specialization, generalizations, or patterns, etc. Using relations. E.g. no_of_sides(rectangle, 4)
6.	PL is a foundation level logic.	FOL is a higher level logic.
7.	PL is not sufficiently expressive to represent complex statements.	FOL can represent complex statements.

Scanned by CamScanner

Sr. No.	Propositional logic (PL)	Predicate logic (FOL)
8.	PL assumes the world contains facts	FOL assumes the world contains objects, relations, functions like natural language.
9.	In PL Meaning of the facts is context-independent unlike natural language.	In FOL Meaning of the sentences is context dependent like natural language.
10.	PL is declarative in nature.	FOL is derivative in nature.

3.8 Inference in FOL

3.8.1 Forward Chaining

- For any type of inference there should be a path from start to goal. When based on the available data a decision is taken, then the process is called as the forward chaining. Forward chaining or data-driven inference works from an initial state, and by looking at the premises of the rules (IF-part), perform the actions (THEN-part), possibly updating the knowledge base or working memory. This continues until no more rules can be applied or some cycle limit is met.
- For example, "If it is raining then, we will take umbrella". Here, "it is raining" is the data and "we will take umbrella" is a decision. This means it was already known that it's raining that's why it was decided to take umbrella. This process is **forward chaining.**

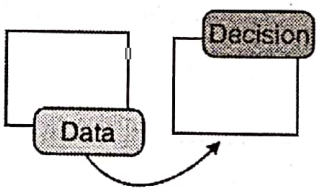


Fig. 3.8.1 : Forward chaining

- "Forward chaining" is called as a data-driven inference technique.

Example

Given :

- **Rule :** human(A) → mortal(A)
- **Data :** human(Mandela)
- **To prove :** mortal(Mandela)

Forward Chaining Solution

- Human (Mandela) matches Left Hand Side of the Rule. So, we can get A = Mandela
- based on the rule statement we can get : mortal(Mandela)
- Forward chaining is used by the "design expert systems", as it performs operation in a forward direction (i.e. from start to the end).

Example

- Consider following example. Let us understand how the same example can be solved using both forward.
- Given facts are as follows:
  1. It is a crime for an American to sell weapons to the enemy of America.
  2. Country Nono is an enemy of America.
  3. Nono has some missiles.

Scanned by CamScanner

4. All the missiles were sold to Nono by Colonel West.
5. Missile is a weapon.
6. Colonel West is American.

- We have to prove that West is a criminal.
- Let's see how to represent these facts by FOL.

1. It is a crime for an American to sell weapons to the enemy nations.
   American(x) ∧ Weapon(y) ∧ sell (x, y, z) ∧ enemy(z, America) => Criminal (x)
2. Country Nono is an enemy of America.
   Enemy (Nono, America)
3. Nono has some missiles.
   - Owns (Nono, x)
   - Missile(x)
4. All the missiles were sold to Nono by Colonel West.
   Missile(x)∧ owns(Nono, x)=> Sell(West, x, Nono)
5. Missile is a weapon.
   Missile(x)=> weapon(x)
6. Colonel West is American.
   American (West)

Proof by forward chaining

The proof will start from the given facts. And as we can derive other facts from those, it will lead us to the solution. Please refer to Fig. 3.8.2 As we observe from the given facts we can reach to the predicate Criminal (West).

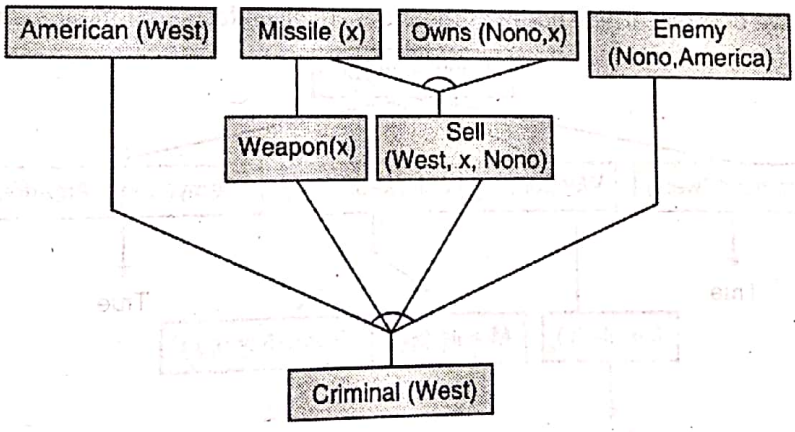


Fig. 3.8.2 : Proof by forward chaining

3.8.2 Backward Chaining

MU – Dec. 12, May 14

Q. Describe backward chaining algorithm with an example. (Dec. 12, 10 Marks)

Q. Explain backward chaining giving suitable example. (May 14, 10 Marks)

- If based on the decision the initial data is fetched, then it is called as **backward chaining**. Backward chaining or goal-driven inference works towards a final state, and by looking at the working memory to see if goal already there. If not look at the actions (THEN-parts) of rules that will establish goal, and set up sub-goals for achieving premises of the rules (IF-part).This continues until some rule can be applied, apply to achieve goal state.

Scanned by CamScanner

- **For example,** If while going out one has taken umbrella. Then based on this decision it can be guessed that it is raining. Here, "taking umbrella" is a decision based on which the data is generated that "it's raining". This process is **backward chaining.**"Backward chaining" is called as a decision-driven or goal-driven inference technique.

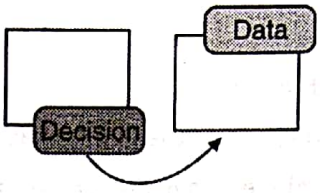


Fig. 3.8.3 : Backward chaining

- **Given :**
  - Rule : human(A) → mortal(A)
  - Data : human(Mandela)
- To prove : mortal(Mandela)

Backward Chaining Solution

- mortal (Mandela) will be matched with mortal (A) which gives human (A) i.e. human (Mandela) which is also a given fact. Hence proved.
- It makes use of right hand side matching. backward chaining is used by the "diagnostic expert systems", because it performs operations in a backward direction (i.e. from end to start).

Example

Let us understand how the same example used in forward chaining can be solved using backward chaining.

Proof by backward Chaining

The proof will start from the fact to be proved. And as we can map it with given facts, it will lead us to the solution. Please refer to Fig. 3.8.4. As we observe, all leaf nodes of the proof are given facts that means "West is Criminal".

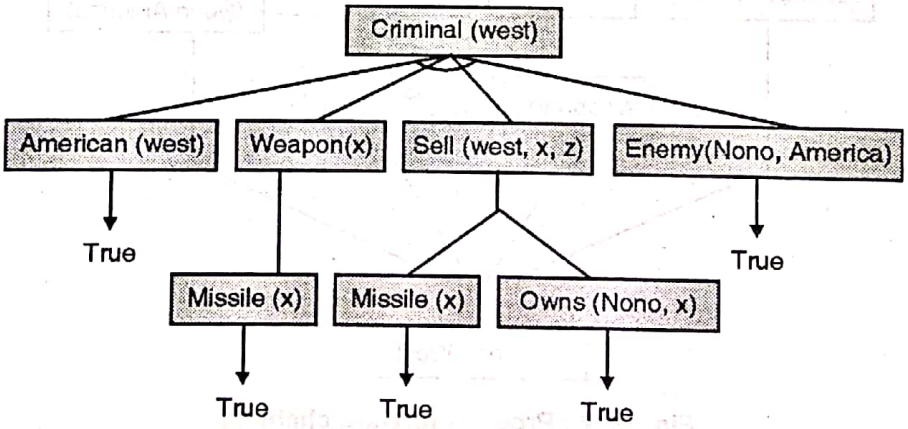


Fig. 3.8.4 : Proof by backward chaining

3.8.3 Differentiate between Forward Chaining and Backward Chaining

Attribute	Backward Chaining	Forward Chaining
Also known as	Goal-driven	Data-driven
Starts from	Possible conclusion	New data
Processing	Efficient	Somewhat wasteful
Aims for	Necessary data	Any Conclusion(s)

Scanned by CamScanner

Attribute	Backward Chaining	Forward Chaining
Approach	Conservative/Cautious	Opportunistic
Practical if	Number of possible final answers is reasonable or a set of known alternatives is available.	Combinatorial explosion creates an infinite number of possible right answers.
Appropriate for	Diagnostic, prescription and debugging application	Planning, monitoring, control and interpretation application
Reasoning	Top-down reasoning	Bottom-up reasoning
Type of Search	Depth-first search	Breadth-first search
Who determine search	Consequents determine search	Antecedents determine search
Flow	Consequent to antecedent	Antecedent to consequent

Ex. 3.8.1 : Using predicate logic find the course of Anish's liking for the following :

(i) Anish only likes easy courses.

(ii) Computer courses are hard.

(iii) All electronics courses are easy

(iv) DSP is an electronics course.

Soln. :

Step 1 : Converting given facts to FOL

(i) $\forall x$: course (x) $\wedge$ easy (x) $\rightarrow$ likes (Anish, x)

(ii) $\forall x$: course (x) $\wedge$ computes (x) $\rightarrow$ hard (x)

(iii) $\forall x$: course (x) $\wedge$ electronics (x) $\rightarrow$ easy (x)

(iv) Electronics (DSP)

(v) course (DSP)

Step 2 : Proof by backward chaining :

As we have to find out which course Anish likes. So we will start the proof from the same fact.

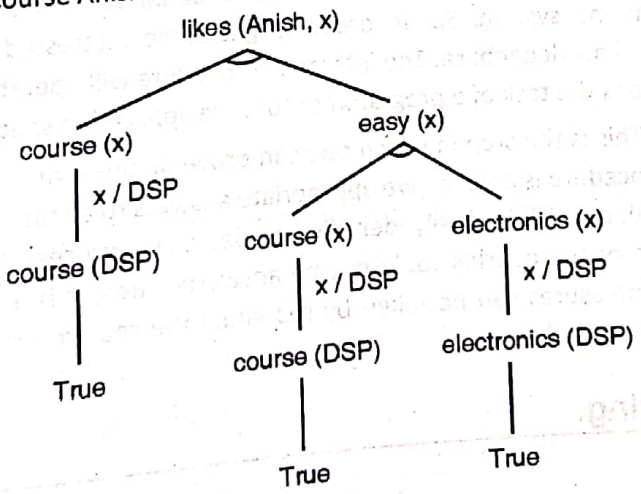


Fig. P. 3.8.1

Hence proved that Anish likes DSP course.

Scanned by CamScanner

3.9 Knowledge Engineering in First Order Logic

Knowledge engineering is a process of knowledgebase construction. It requires a knowledge engineer to investigate a particular domain, learn the important concepts in that domain, and create a formal representation and logical relations among objects in that domain.

3.9.1 Knowledge Engineering Process

Following is the general knowledge engineering process which can be applied to problem of any domain.

1. **Identify the task:** This step is analogues to PEAS process while designing an agent. While identifying task, the knowledge engineer must define the scope of knowledgebase and the range of questions that can be answered through the database. He also need to specify the type of facts that will be available for each specific problem instance.
2. **Assemble the relevant knowledge:** Assembling the relevant knowledge of that particular domain is called the process of knowledge acquisition. In this the knowledge engineer needs to extract the domain knowledge either by himself provided he is the domain expert or needs to work with the real experts of the domain. In this process knowledge engineer learns how the domain actually works and can determine the scope of the knowledgebase as per the identified tasks.
3. **Defining vocabulary:** Defining a complete vocabulary including predicates, functions and constants is a very important step of knowledge engineering. This process transforms the domain level concepts to logic level symbols. It should be exhaustive and precise. This vocabulary is called as ontology of the domain. And once the ontology is defined, it means, the existence of the domain is defined. That is, what kind of things exist in the domain is been decided.
4. **Encoding of general knowledge about the domain:** In this step the knowledge engineer defines axioms for all the vocabulary terms by define meaning of each term. This enables expert to cross check the vocabulary and the contents. If he finds any misinterpretations or gaps, it can be fixed at this point by redoing step 3.
5. **Encode the problem:** In this step, the specific problem instance is encoded using the defined ontology. This step will be very easy if the ontology is defined properly. Encoding means writing atomic sentences about problem instances which are already part of ontology. It can be analogues to input data for a computer program.
6. **Query the Knowledgebase:** Once all the above steps are done, all input for the system is set and now is a time to generate some output from the system. So, in order to get some interested facts inferred from the provided knowledge, we can query the knowledgebase. The inference procedure will operate on the axioms and facts to derive the new inferences. This lessens the task of a programmer to write application specific programs.
7. **Debug the knowledgebase:** This is the step in which one can prove or check the toughness of the knowledgebase. In the sense, if the inference procedure is able to give appropriate answers to all the queries asked or it stops in between because of the incomplete axioms; will be easily identified by debugging process. If one observes the reasoning chain stopping in between or some of the queries could not be answered then, it is an indication of a missing or a weak axioms. Then the corrective measures can be taken by repeating the required steps and system can be claimed to have a complete and precise knowledgebase.

3.10 Unification and Lifting

3.10.1 Unification

The processes of finding legal substitutions that make different logical expressions look identical. The unification algorithm is a recursive algorithm; the problem of **unification** is: given two atoms, to find if they unify, and, if they do, return an MGU (Most General Unifier) of them.

Scanned by CamScanner

```
Procedure Unify(t1,t2)
  Inputs
      t1,t2: atoms Output
      most general unifier of t1 and t2 if it exists or ⊥ otherwise
  Local
      E: a set of equality statements
      S: substitution
  E ← {t1=t2}
  S={}
  while (E≠{})
      select and remove x=y from E
      if (y is not identical to x) then
          if (x is a variable) then
              replace x with y everywhere in E and S
              S ← {x/y}∪S
          else if (y is a variable) then
              replace y with x everywhere in E and S
              S ← {y/x}∪S
          else if (x is f(x1,...,xn) and y is f(y1,...,yn)) then
              E ← E∪{x1=y1,...,xn=yn}
          else
              return ⊥
  return S
```

- Unification algorithm for Data log

Example:"x King(x) ∧ Brave(x) ⇒ Noble(x)

King(Ram)

Brave(Ram)

- We get an 'x' where, 'x' is a king and 'x' is brave (Then x is noble) then ideally what we want is Θ= {substitution set}

i.e. Θ= {x/ Ram}

Hence, Ram Unifies x.

3.10.2 Lifting

The process of encapsulating inference rule is called as **Generalized Modus Ponens.**

Generalized Modus Ponens

- For atomic sentences p_i, p_i', and q, where there is a substitution Θ such that

$$\text{SUBST}(\Theta, p_i) = \text{SUBST}(\Theta, p_i') \quad \text{for all i,}$$

$$\frac{p_1', p_2', p_3', \ldots p_n', (p_1 \wedge p_2 \wedge p_3 \wedge \ldots \ldots p_n \Rightarrow q)}{\text{SUBST}(\Theta, q)}$$

- N + 1 premises = N atomic sentences + one implication.
- Applying SUBST(θ , q) produces the conclusion we seek.

$p_1' = King(Ram)$ $\qquad$ $p_2' = Brave(y)$

Scanned by CamScanner

$p_1 = King(x)$ $p_2 = Brave(x)$

$\Theta = \{x / Ram, y / Ram\}$ $q = Noble(x)$

SUBST(Θ,q) is $Noble(Ram)$

- Generalized Modus Ponens is a **lifted** version of Modus Ponens. It raises Modus Ponens from ground (variable-free) propositional logic to first-order logic. Hence it is called as **lifting**.
- Here "lifted" indicates transformed from.
- The major advantage of lifted inference rules over propositional logic is that only those substitutions are made that are required so as particular inferences are allowed to proceed.

Ex. 3.10.1 : Represent following sentences in FOL using a consistence vocabulary.

(i) Every person who buys a policy is smart.

(ii) No person buys an expensive policy.

(iii) There is an agent who sells policies only to people who are not insured.

(iv) There is a barber who shaves all men in town who do not save themselves.

Soln. :

(i) $\forall x \forall y$: person (x) $\wedge$ policy (y) $\wedge$ buys (x, y) $\rightarrow$ smart (x)

(ii) $\forall x, \forall y$: person (x) $\wedge$ policy (y) $\wedge$ expensive (y) $\rightarrow \sim$ buys (x, y)

(iii) $\forall x$: person (x) $\wedge \sim$ insured (x)

$\forall y : \exists x \wedge$ policy (y) $\sim$ agent (x) $\rightarrow$ sells (x, y, x)

(iv) $\exists x \ \forall y$: barber (x) $\wedge$ person (y) $\wedge \sim$ shaves (y, y) $\rightarrow$ shaves (x, y)

Ex. 3.10.2 : Represent following statements in FOPL.

1. Anyone who kills an animal is loved by no one.
2. A square is breezy if there is a pit in the neighbouring squares.

Soln. :

(i) $\forall x \forall y$: kills (x, animal) $\Rightarrow \sim$ loves (y, x)

(ii) $\forall x \forall y$: pit (x, y) $\rightarrow$breez (x, y – 1) $\wedge$breez (x, y + 1) $\wedge$breez (x – 1, y) $\wedge$breez (x + 1, y)

Ex. 3.10.3 : Write first order logic statements for following statements :

(i) If a perfect square is divisible by a prime p then it is also divisible by square of p.

(ii) Every perfect square is divisible by some prime.

(iii) Alice does not likechemistry and history.

(iv) If it is Saturday and warm, then sam is in the park.

(v) Anything anyone eats and is not killed by is food.

Soln. :

(i) $\forall x$: square(x) $\wedge$ prime (y) $\wedge$ divides (p, x) $\rightarrow$ [$\exists z$: square_of(z, p) $\wedge$ divides(z, x)]

(ii) $\forall x \exists y$: square(x) $\wedge$ divides (p, x)

(iii) $\sim$ likes(Alice, History) $\wedge \sim$ likes (Alice, Chemistry)

(iv) day(Saturday) $\wedge$ weather(warm) $\rightarrow$in_park(Sam)

(v) $\forall x$: $\forall y$: person(x) $\wedge$ eats(x, y) $\wedge \sim$killed (x) $\rightarrow$ food(y)

Scanned by CamScanner

3.11 Resolution

- Resolution is a valid inference rule. Resolution produces a new clause which is implied by two clauses containing complementary literals. This resolution rule was discovered by Alan Robinson in the mid 1960's.
- We have seen that a literal is an atomic symbol or a negation of the atomic symbol (i.e.A, ¬A).
- Resolution is the only interference rule you need, in order to build a sound (soundness means that every sentence produced by a procedure will be "true") and complete (completeness means every "true" sentence can be produced by a procedure) theorem proof maker.
- Take an example where we are given that :
  - A clause X containing the literal : Z
  - A clause Y containing the literal : ¬Z
- Based on resolution and the information given above we can conclude :
  (X – {Z}) U (Y – {¬Z})
- Take a generalized version of the above problem :

Given:

- A clause X containing the literal: Z
- A clause Y containing the literal: ¬Y
- A most general unifier G of Z and ¬Y

We can conclude that :((X – {Z}) U (Y – {¬Y})) | G

3.11.1 The Resolution Procedure

- Let knowledge base be a set of true sentences which do not have any contradictions, and Z be a sentence that we want to prove.
- The Idea is based on the proof by negation. So, we should assume ¬Z and then try to find a contradiction (You must have followed such methods while solving geometry proofs). Then based on the Intuition that, if all the knowledge base sentences are true, and assuming ¬Z creates a contradiction then Z must be inferred from knowledge base. Then we need to convert knowledge base U {¬Z} to clause form.
- If there is a contradiction in knowledge base, that means Z is proved. Terminate the process after that.
- Otherwise select two clauses and add their resolvents to the current knowledge base. If we do not find any resolvable clauses then the procedure fails and then we terminate. Else, we have to start finding if there is a contradiction in knowledge base, and so on.

3.11.2 Conversion from FOL Clausal Normal Form (CNF)

MU - May 16

Q. Explain the steps involved in converting the propositional logic statement into CNF with a suitable example. (May 16, 10 Marks)

1. Elimination of implication i.e. Eliminate all '→' : Replace P →Q with ¬P V Q
2. Distribute negations:Replace ¬¬P with P, ¬(PV Q) with ¬P ∧ ¬Q and so on.

Scanned by CamScanner

3. Eliminate existential quantifiers by replacing with Skolem constants or Skolemfunctions:

   e.g. $\forall X \exists Y(P_1(X,Y) \vee (P_2(X,Y)) \equiv \forall X\ (P_1(X,f(X)) \vee\ (P_2(X,f(X)))$

4. Rename variables to avoid duplicate quantifiers.
5. Drop all universal quantifiers
6. Place expression into Cconjunctive NormalForm.
7. Convert to clauses i.e. separates all conjunctions as separate clause.
8. Rename variables to avoid duplicate clauses.

Ex. 3.11.1 : Convert following propositional logic statement into CNF.
$A \rightarrow (B \rightarrow C)$

MU - Dec. 15, 4 Marks

Soln. :

FOL : $A \rightarrow (B \leftrightarrow C)$

Normalizing the given statement.

(i) $A \rightarrow (B \rightarrow C \wedge C \rightarrow B)$

(ii) $(A \rightarrow (B \rightarrow C)) \wedge (A \rightarrow (C \rightarrow B))$

Converting to CNF.

Applying Rule, $\alpha \rightarrow \beta \equiv \sim\alpha \vee \beta$

$\sim A \vee (\sim B \vee C) \wedge \sim A \vee (\sim C \vee B)$

i.e. $\sim A \vee ((\sim B \vee C) \wedge (\sim C \vee B))$

3.11.3 Facts Representation

- To show how facts can be represented let's take a simple problem:
  - "Heads X wins, Tails Y loses."
  - Our goal is to show that X always wins with the help of resolution.
- Solution can be given as follows:
  1. $H \Rightarrow Win(X)$
  2. $T \Rightarrow Loose(Y)$
  3. $\neg H \Rightarrow T$
  4. $Loose(Y) \Rightarrow Win(X)$

  Thus we have : Win(X)
- We can write a proof for this problem as follws:
  1. $\{\neg H, Win(X)\}$
  2. $\{\neg T, Loose(Y)\}$
  3. $\{H, T\}$
  4. $\{\neg Loose(Y), Win(X)\}$

Scanned by CamScanner

5. {¬ Win(X)}
6. {¬T, Win(X)} ..... (From 2 and 4)
7. {T, Win(X)} ..... (From 1 and 3)
8. {Win(X)} ..... (From 6 and 7)
9. {} ..... (From 5 and 8)

3.11.4 Example

Let's take the same example of forward and backward chaining to learn how to write proofs for resolution.

Step 1 :

The given facts are :

1. It is a crime for an American to sell weapons to the enemy nations.

   American(x) ∧ Weapon(y) ∧ sell (x,y,z) ∧ enemy(z, America) => Criminal (x)

2. Country Nono is an enemy of America.

   Enemy (Nono, America)

3. Nono has some missiles.
   - Owns (Nono, x)
   - Missile(x)

4. All the missiles were sold to Nono by Colonel West.

   Missile(x)∧ owns(Nono, x)=> Sell(West, x, Nono)

5. Missile is a weapon.

   Missile(x)=> weapon(x)

6. Colonel West is American.

   American (West)

Step 2 :

Lets convert them to CNF

1. ~ American(x) V ~Weapon(y) V~ sell (x,y,z) V ~enemy(z, America)

   V Criminal (x)

2. Enemy (Nono, America)
3. Owns (Nono, x)
4. Missile(x)
5. ~ Missile(x) V ~ owns(Nono, x) V Sell(West, x, Nono)
6. ~Missile(x) V weapon(x)
7. American (West)

Scanned by CamScanner

Step 3 :

To prove that West is criminal using resolution.

~ criminal (West)

~ American (x) ∨ ~ weapon (y) ∨ ~ sell (x, y, z) ∨ ~ enemy (z, America) ∨ criminal (x)

x/West

~ American (West) ∨ ~ weapon (y) ∨ ~ sell (West, y, z) ∨ ~ enemy (z, America)

enemy (Nono, America)

z/Nono

~ American (West) ∨ ~ weapon (y) ∨ ~ sell (West, y, Nono)

American (West)

~ weapon (y) ∨ ~ sell (West, y, Nono)

~ Missile (x) ∨~ owns (Nono, x) ∨ Sell (West, x, Nono)

~ weapon (y) ∨ ~ Missile (y) ∨ ~ owns (Nono, y)

Missile (x)

~ weapon (y) ∨ owns (Nono, y)

Weapon (x) ∨ ~ Missile (x)

~ Missile (y) ∨ ~ owns (Nono, y)

Missile (x)

~ owns (Nono, y)

owns (Nono, x)

NIL

Hence our assumption was wrong. Hence proved that West is criminal.

Ex. 3.11.2 : Consider following statements :

(a) Ravi Likes all kind of food.

(b) Apple and Chicken are food

(c) Anything anyone eats and is not killed is food.

(d) Ajay eats peanuts and still alive.

(e) Rita eats everything that Ajay eats.

Prove that Ravi Likes Peanuts using resolution. What food does Rita eat?

Soln. :

(A) Proof by Resolution

Step 1 : Negate the statement to be proved.

~likes (Ravi, Peanuts)

Step 2 : Convert given facts to FOL

(a) ∀ x, food (x) ⇒ likes (Ravi, x)

(b) food (Apple)

Scanned by CamScanner

(c) food (Chicken)

(d) $\forall$ x $\forall$ y : eats (x, y) $\wedge$~ killed (x) $\rightarrow$ food (y)

(e) eats (Ajay, Peanuts) $\wedge$ alive (Ajay)

(f) $\forall$ x : eats (Ajay, x) $\rightarrow$ eats (Rita, x)

In this case we have to add few common sense predicate which are always true.

(g) $\forall$ x : ~ killed (x) $\rightarrow$ alive (x)

(h) $\forall$ x : alive (x) $\rightarrow$~ killed (x)

Step 3 : Converting FOLs to CNF

(a) ~food (x) $\vee$ likes (Ravi, x)

(b) food (Apple)

(c) food (Chicken)

(d) ~eats (x, y) $\vee$ killed (x) $\vee$ food (y)

(e) eats (Ajay, Peanuts)

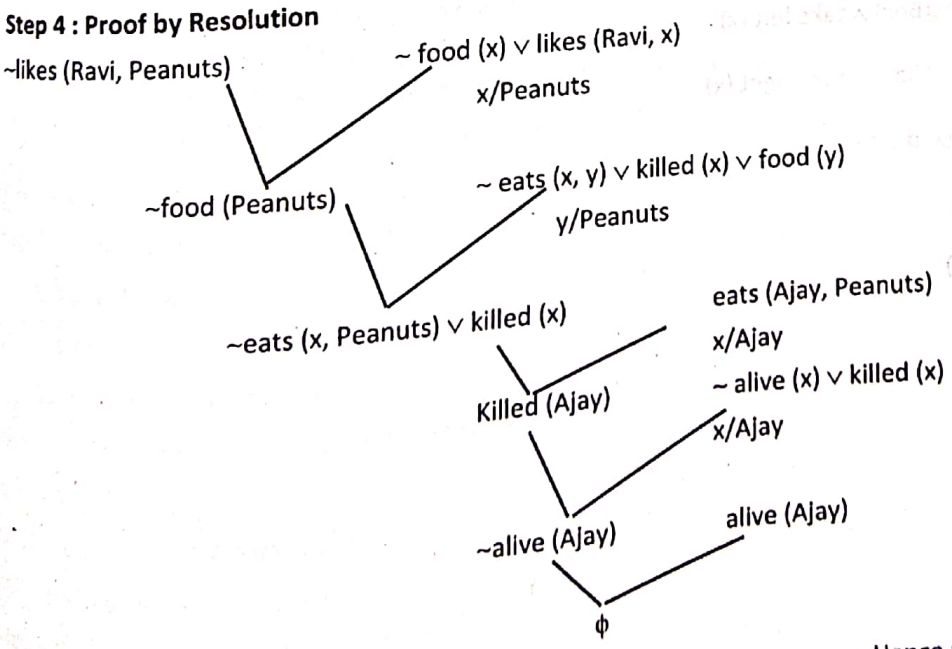
(f) alive (Ajay)

(g) ~eats (Ajay, x) $\vee$ eats (Rita, x)

(h) killed (x) $\vee$ alive (x)

(i) ~alive (x) $\vee$ killed (x)

Step 4 : Proof by Resolution

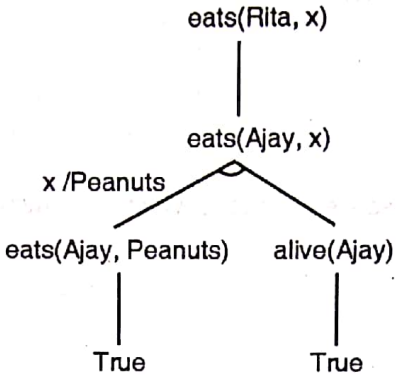


As the result of this resolution is NIL, it means out assumption is wrong. Hence proved that "Ravi likes Peanuts".

Scanned by CamScanner

To answer : What food Rita eats ?

(B) Proof by backward chaining : (Referring to FOLs of Step 2)



Hence the answer is Rita eats peanuts.

Ex. 3.11.3 : Using a predicate logic convert the following sentences to predicates and prove that the statement "Ram did not jump" is false.

(a) Ram went to temple.

(b) The way to temple is, walk till post box and take left or right road.

(c) The left road has a ditch.

(d) Way to cross the ditch is to jump

(e) A log is across the right road.

(f) One needs to jump across the log to go ahead.

Soln.:

Step 1 : Negate the statement to be proved.

~jump (Ram)

Step 2 : Converting given statement to FOL

(a) At (Ram, temple)

(b_1) $\forall$ x : At (x, temple) $\longrightarrow$ At (x, PostBox) $\wedge$ take left (x)

(b_2) $\forall$ x : At (x, temple) $\longrightarrow$ At (x, PostBox) $\wedge$ take right (x)

(c) $\forall$ x: take left (x) $\longrightarrow$ cross (x, ditch)

(d) $\forall$ x cross (x, ditch) $\longrightarrow$ jump (x)

(e) $\forall$ x : take right (x) $\longrightarrow$ at (x, log)

(f) $\forall$ x : at (x, log) $\longrightarrow$ jump (x)

Step 3 : Converting FOLs to CNF

(a) At (Ram, temple)

(b_{11}) ~ At (x, temple) $\vee$ At (x, PostBox)

(b_{12}) ~ At (x, temple) $\vee$ take left (x)

(b_{21}) ~ At (x, temple) $\vee$ At (x, PostBox)

(b_{22}) ~ At (x, temple) $\vee$ take right (x)

Scanned by CamScanner

(c) ~take left (x) ∨ cross (x, ditch)

(d) ~ cross (x, ditch) ∨ jump (x)

(e) ~take right (x) ∨ at (x, log)

(f) ~at (x, log) ∨ jump (x)

Step 4 : Proof by Resolution

~jump (Ram) at (x, log) ∨jump (x)

x/Ram

~ At (Ram, log) ~ take right (x) ∨ at (x, log)

x/Ram

~take right (Ram) ~ At (x, temple) ∨ take right (x)

~ At (x, temple) At (Ram, temple)

ϕ

Hence proved.

Ex. 3.11.4: Consider following statements.

1. Rimi is hungry.
2. If Rimi is hungry she barks.
3. If Rimi is barking then Raja is angry.

Explain statements in predicate logic. Convert them into CNF form.

Prove that Raja is angry using resolution.

Soln. :

Step 1 : Converting given facts to FOL.

1. Hungry (Rimi)
2. Hungry (Rini) → barks (Rimi)
3. Barks (Rimi) → angry (Raja)

Step 2 : Converting FOL statements to CNF.

1. Hungry (Rimi)
2. ~ hungry (Rimi) ∨ barks (Rimi)
3. ~barkes (Rimi) ∨ angry (Raja)

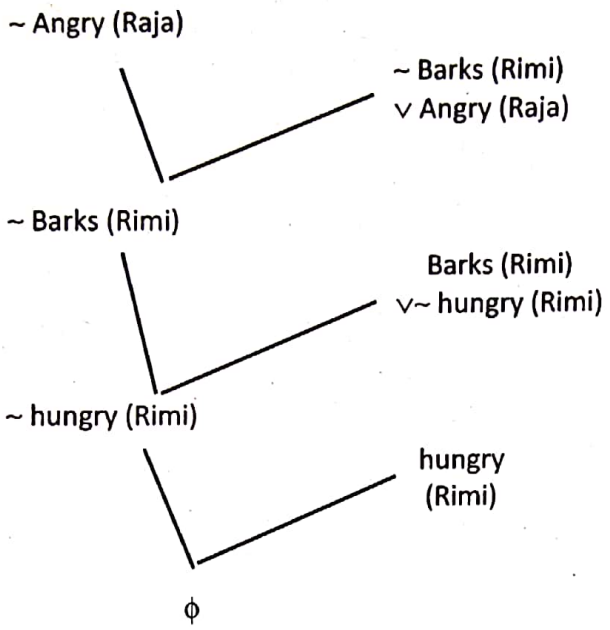
Step 3 : Negate the stmt to be proved

T.P.T. Angry (Raja)

Negation : ~ Angry (Raja)

Scanned by CamScanner

Step 4 : Proof by resolution



This shows that our assumption is Wrong. Hence proved that **Raja is Angry.**

Ex. 3.11.5 : Consider following facts.

1. If maid stole the jewellery then butler was not guilty.
2. Either maid stole the jewellery or she milked the cow.
3. If maid milked the cow then butler got the cream.
4. Therefor if butler was guilty then he got the cream.

Prove the conclusion (step 4) is valid using resolution.

Soln. :

Step 1 : Converting given facts to FOL.

1. steal (maid, jwellary) →~ guilty (butler)
2. steal (mail, jwellary) ∨ milk (maid, cow)
3. silk (maid, cow) →got_Cream (butter)

To prove that

4. guilty (butler) →got_cream (butter)

Step 2 : Converting FOL to CNF.

1. ~ steal (maid, jwellary) ∨~ guilty (butler)
2. steal (maid, jwellary) ∨ milk (maid, cow)
3. ~ milk (maid, cow) ∨got_cream (butler)
4. ~ guilty (butler) ∨got_cream (butler)

Step 3 : Negate the proof sentence

As sentence 4 is the one to be proved.

guilty (butler) ∧~ got_ cream (butler)

Scanned by CamScanner

Step 4 : Proof by resolution $\wedge \vee$ got_cream (butter)

guilty (butler)

~ milk (maid, cow)
$\vee$ got_cream (butler)

guilty (butler) $\vee$ ~ milk (maid, cow)

steal (maid jwellery)
$\vee$ ~ milk (mail, cow)

guilty (butler) $\vee$ steal (maid, jewellery)

~ guilty (butler)
$\vee$ ~ steal (maid jwellery)

ϕ

Hence proved.

Ex.3.11.6 : Consider following axioms.

All people who are graduating are happy.

All happy people smile.

Someone is graduating.

(i) Represent these axioms in FOL.

(ii) Convert each formula to CNF.

(iii) Prove that someone is smiling using resolution technique. Draw the resolution tree.

MU - Dec. 15, 12 Marks

Soln. :

Step 1 : Converting axioms to FOL.

(i) $\forall x$: graduating (x) $\rightarrow$ happy (x).

(ii) $\forall x$: happy (x) $\rightarrow$ smile (x)

(iii) $\exists x$: graduating (x)

Step 2 : Converting FOL to CNF.

(i) ~ graduating (x) $\vee$ happy (x)

(ii) ~ happy (x_1) $\vee$ smile (x_1)

(iii) graduating (x_2)

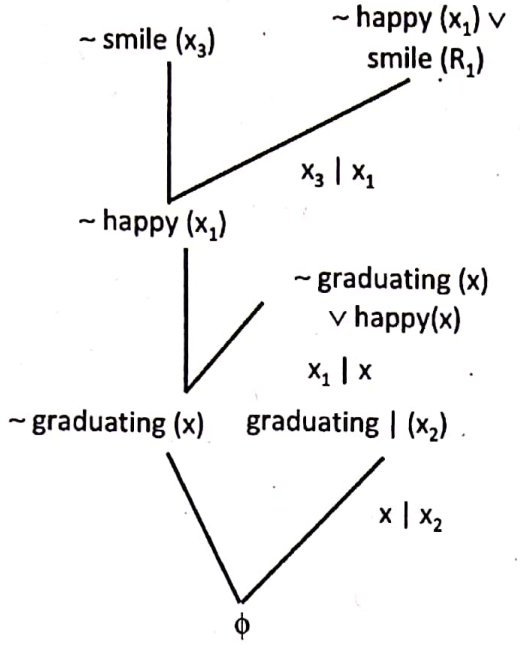
Step 3 : T.P.T. x_3 smile (x_3)

Negating the stmt

~ smile (x_3)

Proof by resolution

Scanned by CamScanner



Hence our assumption is wrong.

Hence proved.

3.12 Planning

- Planning various tasks is a part of day to day activities in real world. Say, you have tests of two different subjects on one day then, you will plan your study timetable as per your strengths and weaknesses in those two subjects.
- Also you must have learnt about various scheduling algorithms (e.g. first in first out) in operating systems subject and how printers plan/schedule their printing tasks based on tasks importance while printing.
- These examples illustrate how planning is important. We have seen that artificially intelligent systems are rational systems. So, devising a plan to perform actions becomes a part of creating an artificially intelligent agent. When we think about giving intelligence to a system or device, we have to make sure that it priorities between given activities or tasks.
- In this we are going to learn about how the machines can become more intelligent by using planning while performing various actions.

3.12.1 Introduction to Planning

Q. What is planning? How it defers from searching ? MU - Dec. 13 (Dec. 13, 10 Marks)

- **Planning** in Artificial Intelligent can be defined as a problem that needs decision making by intelligent systems to accomplish the given target.
- The intelligent system can be a robot or a computer program.
- Take example of a driver who has to pick up and drop people from one place to another. Say he has to pick up two people from two different places then he has to follow some sequence, he cannot pick both passengers at same time.
- There is one more definition of planning which says that, **Planning** is an activity where agent has to come up with a sequence of actions to accomplish target.
- Now, let us see what information is available while formulating a planning problem and what results are expected.
- We have information about the initial status of the agent, goal conditions of agent and set of actions an agent can take.

Scanned by CamScanner

- Aim of an agent is to find the proper sequence of actions which will lead from starting state to goal state and produce an efficient solution.

3.12.2 Simple Planning Agent

- Take example of an agent which can be a coffee maker, a printer and a mailing system, also assume that there are 3 people who have access to this agent.
- Suppose at same time if all 3 users of an agent give a command to execute 3 different tasks of coffee making, printing and sending a mail.
- Then as per definition of planning, agent has to decide the sequence of these actions.

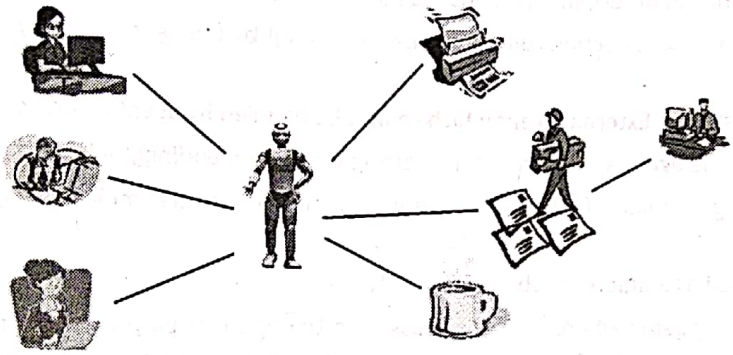


Fig. 3.12.1 : Example of a Planning Problem

- Fig. 3.12.2 depicts a general diagrammatic representation of a planning agent that interacts with environment with its sensors and effectors/actuators. When a task comes to this agent it has to decide the sequence of actions to be taken and then accordingly execute these actions.

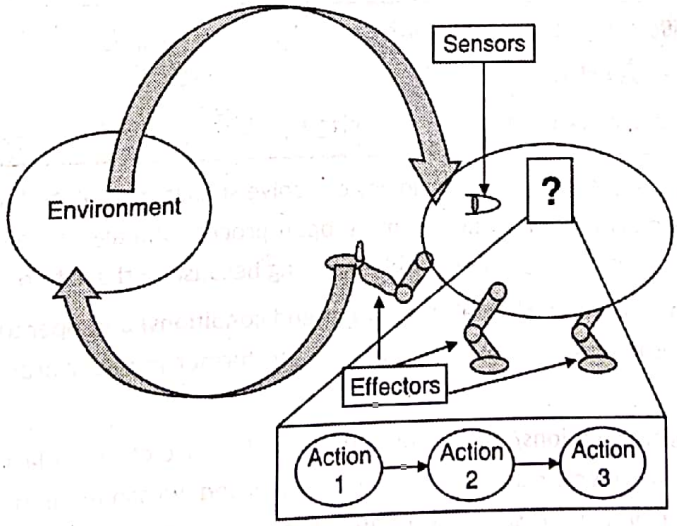


Fig. 3.12.2 : Planning agent

3.13 Planning Problem

MU - Dec. 12

(Dec. 12, 2.5 Marks)

Q. What is planning problem ?

- We have seen in above section what information is available while formulating a planning problem and what results are expected. Also it is understandable here that, states of an agent correspond to the probable surrounding environments while the actions and goals of an agent are specified based on logical formalization.

Scanned by CamScanner

- Also we have learnt about various types of intelligent agents in chapter 1. Which shows that, to achieve any goal an agent has to answer few questions like "what will be the effect of its actions", "how it will affect the upcoming actions", etc. This illustrates that an agent must be able to provide a proper reasoning about its future actions, states of surrounding environments, etc.
- Consider simple Tic-Tac-Toe game. A Player cannot win a game in one step, he/she has to follow sequence of actions to win the game. While taking every next step he/she has to consider old steps and has to imagine the probable future actions of an opponent and accordingly make the next move and at the same time he/she should also consider the consequences of his/her actions.
- **A classical planning has the following assumptions about the task environment:**
  - **Fully Observable :**Agent can observe the current state of the environment.
  - **Deterministic:** Agent can determine the consequences of its actions.
  - **Finite:** There are finite set of actions which can be carried out by the agent at every state in order to achieve the goal.
  - **Static :** Events are steady. External event which cannot be handled by agent is not considered.
  - **Discrete :**Events of the agent are distinct from starting step to the ending(goal) state in terms of time.
- So, basically a planning problem finds the sequence of actions to accomplish the goal based on the above assumptions.
- Also goal can be specified as a union of sub-goals.
- Take example of ping pong game where, points are assigned to opponent player when a player fails to return the ball within the rules of ping-pong game. There can a best 3 of 5 matches where, to win a match you have to win 3 games and in every game you have to win with a minimum margin of 2 points.

3.13.1 Problem Solving and Planning

MU - Dec. 12, Dec. 13, Dec. 15

Q.	How planning problem differs from search problem?	**(Dec. 12, 2.5M)**
Q.	How planning defers from searching ?	**(Dec. 13, 5 Marks)**
Q.	Compare and contrast problem solving agent and planning agent.	**(Dec. 15, 5 Marks)**

- Generally problem solving and planning methodologies can solve similar type of problems. Main difference between problem solving and planning is that planning is a more open process and agents follow logic-based representation. Planning is supposed to be more powerful than problem solving because of these two reasons.
- Planning agent has situations (i.e. states), goals (i.e. target end conditions) and operations (i.e. actions performed).All these parameters are decomposed into sets of sentences and further in sets words depending on the need of the system.
- Planning agents can deal with situations/states more efficiently because of its explicit reasoning capability also it can communicate with the world. Agents can reflect on their targets and we can minimize the complexity of the planning problem by independently planning for sub-goals of an agent. Agents have information about past actions, presents actions and the important point is that it can predict the effect of actions by inspecting the operations.
- Planning is a logical representation, based on situation, goals and operations, of problem solving.

Planning = Problem solving + Logical representation

3.14 Goal of Planning

- We plan activities in order to achieve some goals. Main goal can be divided into sub-goals to make planning more efficient.

Scanned by CamScanner

- Take example of a grocery shopping at supermarket, suppose you want to buy milk, bread and egg from supermarket, then your initial state will be – "at home" and goal state will be – "get milk, bread and egg".
- Now if you look at the Fig. 3.14.1 you will understand that branching factor can be enormous depending upon the set of actions, for e.g. Watch TV, read book, etc., available at that point of time.

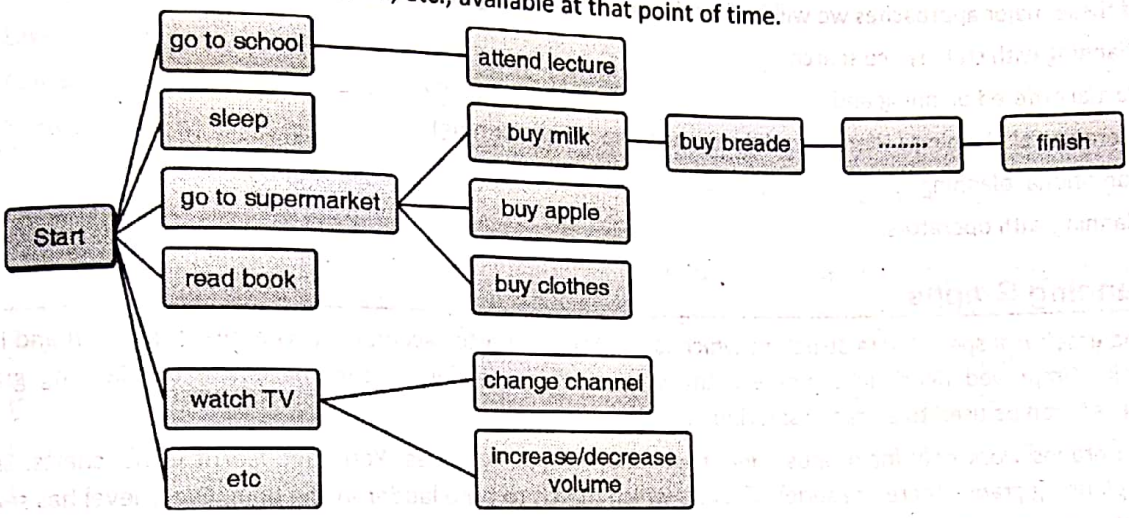


Fig. 3.14.1 : Supermarket examples to understand need of planning

- Thus **branching factor** can be defined as a set of all probable actions at any state, set can be very large, such as in the supermarket example or block problem. If the domain of probable actions increases, the branching factor will also increase as they are directly proportional to each other, this will result in the increase of search space.
- To reach the goal state you have to follow many steps, if you consider using heuristic functions, then you have to remember that it will not be able to eliminate states; these functions will be helpful only for guiding the search of states.
- So it becomes difficult to choose best actions. (i.e. even if we go to supermarket we need to make sure that all three listed items are picked, only then goal state can be achieved).
- As there are many possible actions and it is difficult to describe every state, and there can be combined goals (as seen in supermarket example) searching is inadequate to achieve goals efficiently. In order to be more efficient planning is required.
- In above sections, we discussed that planning requires explicit knowledge that means in case of planning we need to know the exact sequence of actions which will be useful in order to achieve the goal.
- Advantage of planning is that, the order of planning and the order of execution need not be same. For example, you can plan how to pay bills for grocery before planning to go to supermarket.
- Another advantage of planning is that you can make use of divide and conquer policy by dividing /decomposing the goal into sub goals.

3.14.1 Major Approaches

- There are many approaches to solve planning problems. Following are few **major approaches** used for planning :
  - Planning with state space search.
  - Partial ordered planning.
  - Hierarchical planning / hierarchical decomposition (HTN planning).
  - Planning situation calculus/ planning with operators.
  - Conditional planning.
  - Planning with operators.

Scanned by CamScanner

- Planning with graphs.
- Planning with propositional logic.
- Planning reactive.

– Out of these major approaches we will be learning about following approaches in detail :
  - Planning with state space search
  - Partial ordered planning and
  - Hierarchical planning / Hierarchical decomposition (HTN planning).
  - Conditional planning.
  - Planning with operators.

3.15 Planning Graphs

– Planning graph is a special data structure which is used to get better accuracy. It is a directed graph and is useful to accomplish improved heuristic estimates. Any of the search technique can make use of planning graphs. Also GRAPHPLAN can be used to extract a solution directly.

– Planning graphs work only for propositional problems without variables. You have learnt Gantt charts. Similarly, in case of planning graphs there are series of levels which match to time ladder in the plan. Every level has set of literals and a set of actions. Level 0 is the initial state of planning graph.

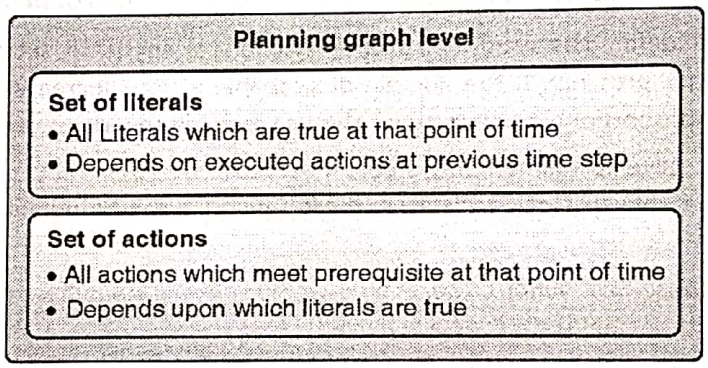


Fig. 3.15.1 : Planning graph level

Example

– Init(Have(Apple))
– Goal(Have(Apple) ∧Ate(Apple))
– Action(Eat(Apple), PRECOND: Have(Apple)
– EFFECT: ¬Have(Apple) ∧Ate(Apple))
– Action(Cut(Apple), PRECOND: ¬ Have(Apple)
– EFFECT: Have(Apple))

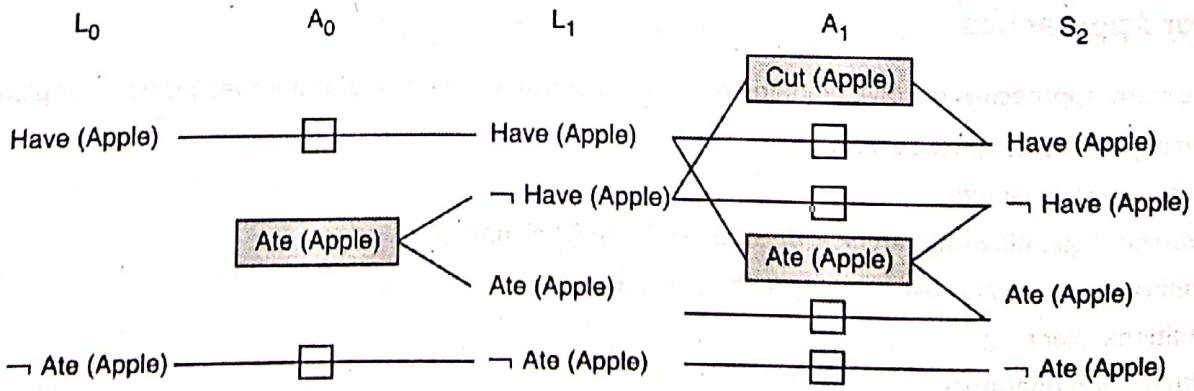


Fig. 3.15.2

Scanned by CamScanner

- Start at level L0 and determine action level A0 and next level L1.
  - A0 >> all actions whose prerequisite is satisfied at previous level.
  - Connect precondition and effect of actions L0 – >L1.
  - Inaction is represented by *persistence actions*.
  - Level A0 contains the possible actions.
  - Conflicts between actions are shown by *mutual exclusion* links.
  - Level L1 contains all literals that could result from picking any subset of actions in Level A0.
  - Conflicts between literals which cannot occur together (as a effect of selection action) are represented by mutual exclusion links.
  - L1 defines multiple states and the mutual exclusion links are the constraints that define this set of states.
  - Continue until two consecutive levels are the same or contain the same amount of literals.

A mutual exclusion relation holds between two actions when :	A mutual exclusion relation holds between two literals when :
– One action cancels out the effect of another action. – One of the effects of action is negation of preconditions of other action. – One of the preconditions of one action is mutually exclusion with the precondition of the other action.	– If one literal is the negation of the other literal OR. – If each possible action pair that could achieve the literal is mutually exclusive.

GRAPH PLAN algorithm

GRAPHPLAN can directly extract solution from planning graph with the help of following algorithm :

```
function GRAPHPLAN(problem) return solution or failure
    graph ← INITIAL-PLANNING-GRAPH(problem)
    goals ← GOALS[problem]
    loop do
        if goals all non-mutex in last level of graph then do
        solution ← EXTRACT-SOLUTION(graph, goals,
LENGTH(graph))
        if solution ≠ failure then return solution
        else if NO-SOLUTION-POSSIBLE(graph) then return failure
    graph ← EXPAND-GRAPH(graph, problem)
```

Properties of planning graph

- If goal is absent from last level then goal cannot be achieved!.
- If there exists a path to goal then goal is present in the last level.
- If goal is present in last level then there may not exist any path.

3.16 Planning as State-Space Search

- We have seen example of an agent that can perform three tasks of printing, sending a mail and making coffee namely lets' call this agent as office agent.
- When this office agent gets order from three people at a same time to perform these three different tasks then, let us see how planning with state space search problem will look if we have a finite space.

Scanned by CamScanner

- You can understand from Fig. 3.16.1 that the office agent is at location 250 on the state space grid. When he gets a task he has to decide which task can be performed more efficiently in lesser time.

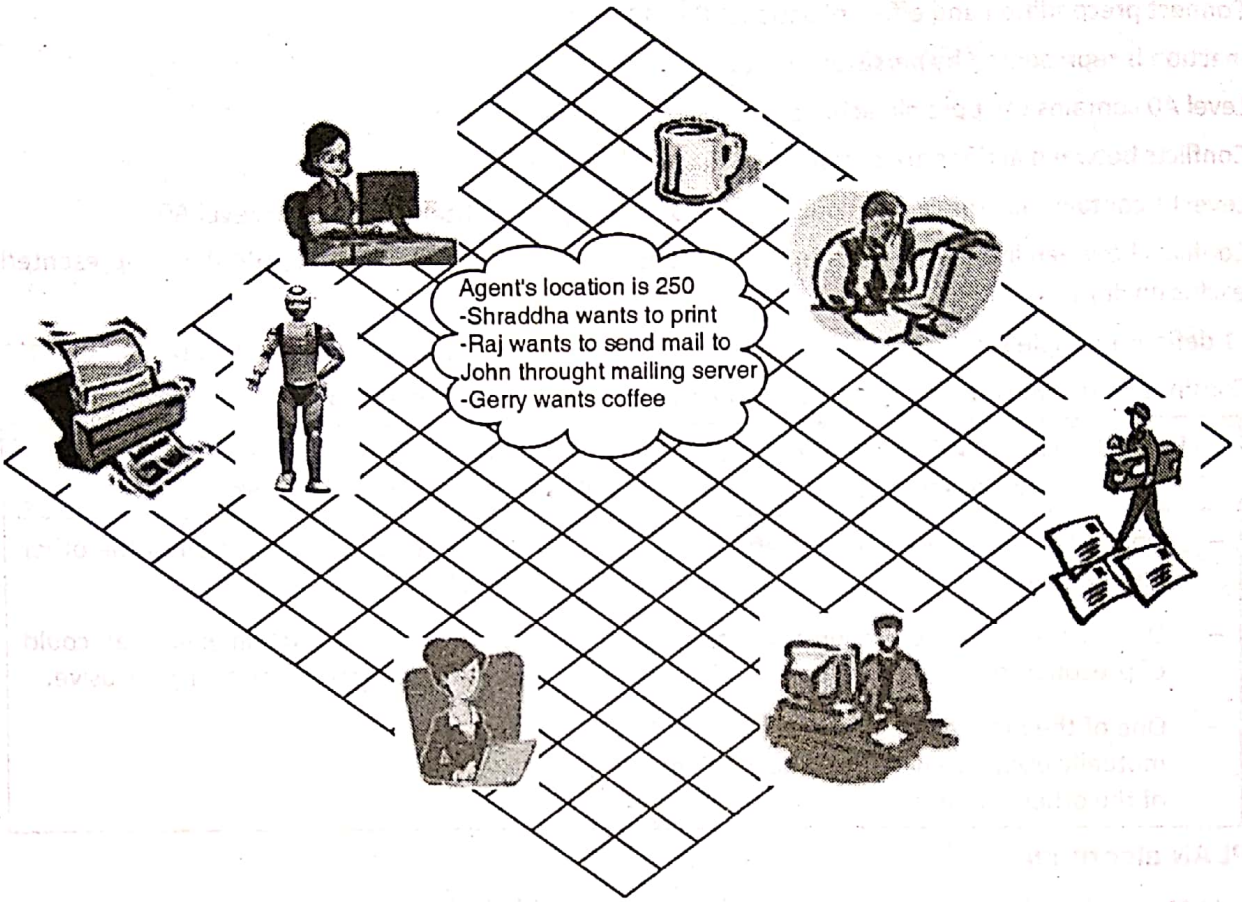


Fig. 3.16.1 : Office agent example with finite state space

- If it finds some input and output locations nearer on state space grid for example in case of printing task then the probability of performing that task will increase.
- But to do this it should be aware of its own current location, the locations of people who are assigning tasks and the locations of the required devices.
- State space search is unfavourable for solving real-world problems because, it requires complete description of every searched state, also search should be carried out locally.
- There can be two ways of representations for a state:

1. Complete world description
2. Path from an initial state

1. Complete world description

- Description is available in terms of an assignment of a value to each previous suggestion.
- Or we can say that description is available as a suggestion that defines the state.
- Drawback of this types is that it requires a large amount of space.

2. Path from an initial state

- As per the name, path from an initial state gives the sequence of actions which are used to reach a state from an initial state.
- In this case, what holds in a state can be deduced from the axiom which specifies the effects of an actions.

Scanned by CamScanner

- Drawback of these types is that it does not explicitly specify "What holds in every state". Because of this it can be difficult to determine whether two states are same.

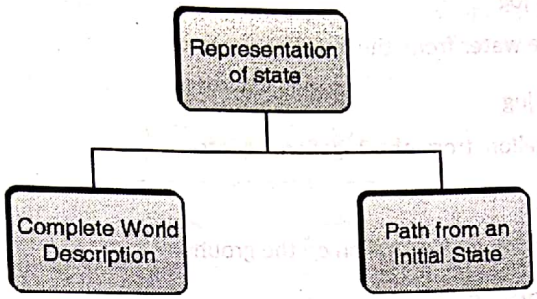


Fig. 3.16.2 : Representation of states

3.16.1 Example of State Space Search

Let us take an example of a water jug problem

- We have two jugs, a 4- gallon one and a 3- gallon one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water how can you get exact 2 gallons of water into the 4- gallon jug?
- The state space for this problem can be described as of ordered pairs of integers (x, y), such that x = 0,1,2,3 or 4, representing the number of gallons of water in the 4- gallon jug and y = 0,1,2 or 3, representing the quantity of water in the 3- gallon jug. The start state is (0, 0). The goal state is (2, n) for any value of n, since the problem does not specify how many gallons need to be in the 3- gallon jug.
- The operators to be used to solve the problem can be described as shown bellow. They are represented as rules whose left side are matched against the current state and whose right sides describe the new state that results from applying the rule.

Rule set

1. (x,y) ⟶ (4,y) fill the 4- gallon jug

   If x<4
2. (x,y) ⟶ (x,3) fill the 3-gallon jug

   If x<3
3. (x,y) ⟶ (x-d,y) pour some water out of the 4- gallon jug

   If x>0
4. (x,y) ⟶ (x-d,y) pour some water out of the 3- gallon jug

   If y>0
5. (x,y) ⟶ (0,y) empty the 4- gallon jug on the ground

   If x>0
6. (x,y) ⟶ (x,0) empty the 3- gallon jug on the ground

   If y>0
7. (x,y) ⟶ (4,y-(4-x)) pour water from the 3- gallon jug into the 4- gallon

   If x+y>=4 and y>0 jug until the 4-galoon jug is full
8. (x,y) ⟶ (x-(3-y),3)) pour water from the 4- gallon jug into the 3-gallon

   If x+y>=3 and x>0 jug until the 3-gallon jug is full

Scanned by CamScanner

9. (x,y) ⟶ (x+y,0) pour all the water from the 3 -gallon jug into

If x+y<=4 and y>0 the 3-gallon jug

10. (x,y) ⟶ (0,x+y) pour all the water from the 4 -gallon jug into

If x+y<=3 and x>0 the 3-gallon jug

11. (0,2) ⟶ (2,0) pour the 2-gallon from the 3 -gallon jug into

the 4-gallon jug

12. (2,y) ⟶ (0,x) empty the 2 gallon in the 4 gallon on the ground.

Production for the water jug problem

Gallons in the 4- gallon Jug	Gallons in the 3- gallon	Rule Applied
0	0	
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5 or 12
2	0	9 or 11

One solution to the water jug problem.

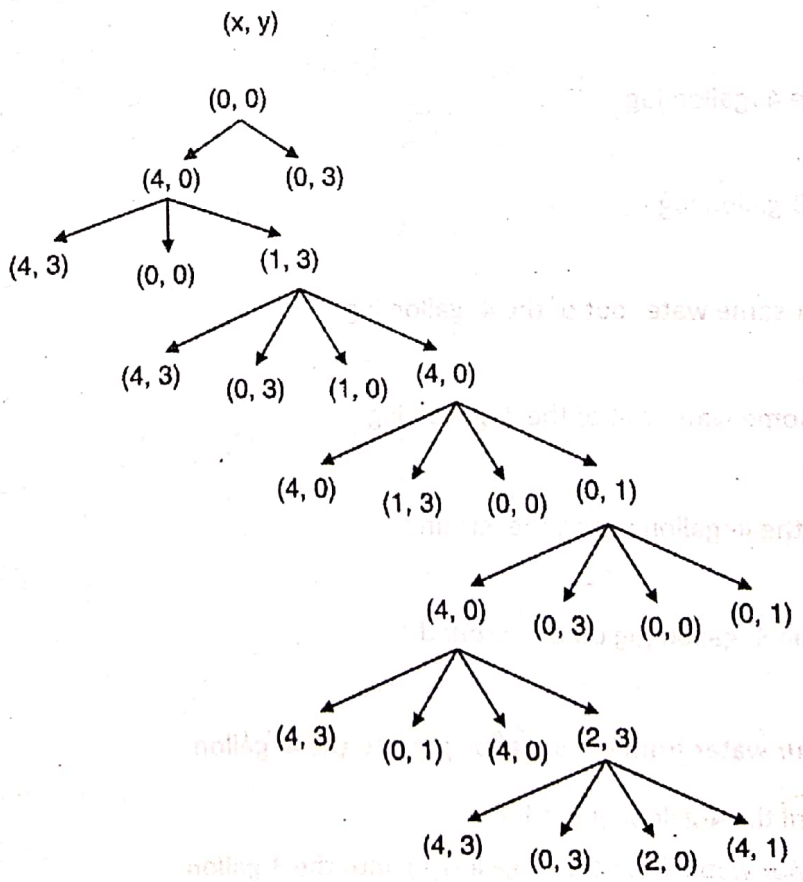


Fig. 3.16.3

Scanned by CamScanner

3.17 Classification of Planning with State Space Search

- As the name suggests state space search planning techniques is based on the spatial searching.

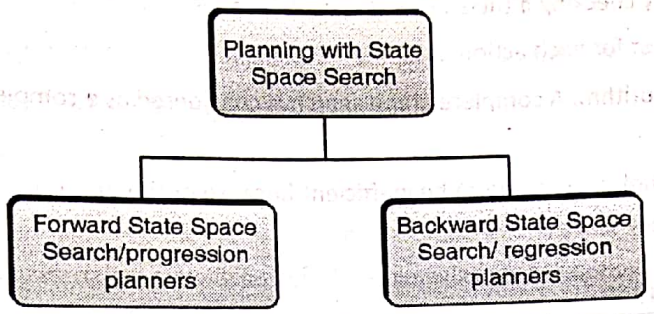


Fig. 3.17.1: Classification of Planning with State Space Search

- Planning with state space search can be done by both forward and backward state-space search techniques.

3.18 Progression Planners

- "Forward state-space search" is also called as **"progression planner".** It is a deterministic planning technique, as we plan sequence of actions starting from the initial state in order to attain the goal.
- With forward state space searching method we start with the initial state and go to final goal state. While doing this we need to consider the probable effects of the actions taken at every state.
- Thus the prerequisite for this type of planning is to have initial world state information, details of the available actions of the agent, and description of the goal state.
- Remember that details of the available actions include preconditions and effects of that action.
- See Fig. 3.18.1 it gives a state-space graph of progression planner for a simple example where flight 1 is at location A and flight 2 is also at location A. these flights are moving from location A to location B. In 1st case only flight 1 moves from location A to location B, so the resulting state shows that after performing that action flight 1 is at location B where as flight 2 is at its original location – A. similarly In 2nd case only flight 2 moves from location A to location B and the resulting state shows that after performing that action flight 2 is at location B while flight 1 is at its original location – A.

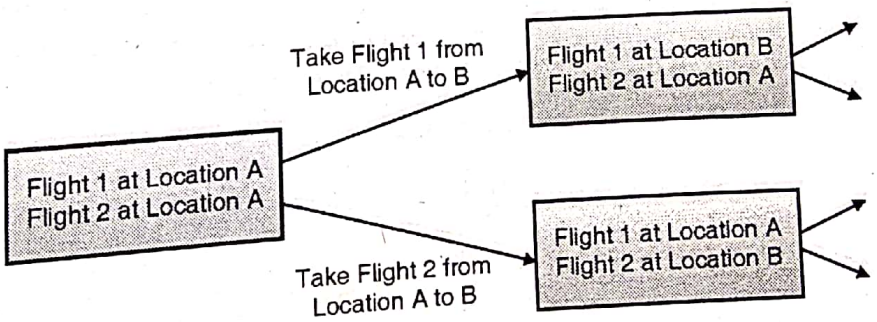


Fig. 3.18.1 : State-space graph of a Progression planners

- It can be observed from the Fig. 3.18.1 that, rectangles show state of the flights (i.e. their current location), and lines give the corresponding actions from one state to another (i.e. move from one location to other location).
- Here the lines coming out of every state matches to all of the permissible actions which can be accepted if the agent is in that state.

Progression planner algorithm

1. Formulize the state space search problem :
   - Initial state is the first state of the planning problem which has a set of positive, the literals which don't appear are considered as false.

Scanned by CamScanner

- If preconditions are satisfied then the actions are favoured i.e. if the preconditions are satisfied then positive effect literals are added for that action else the negative effect literals are deleted for that action.
- Perform goal testing by checking if the state will satisfy the goal.
- Lastly keep the step cost for each action as 1.

2. Consider example of A* algorithm. A complete graph search is considered as a complete planning algorithm. Functions are not used.
3. Progression planner algorithm is supposed to be inefficient because of the irrelevant action problem and requirement of good heuristics for efficient search.

3.19 Regression Planners

- "Backward state-space search" is also called as **"regression planner"** from the name of this method you can make out that the processing will start from the finishing state and then you will go backwards to the initial state.
- So basically we try to backtrack the scenario and find out the best possibility, in-order to achieve the goal to achieve this we have to see what might have been correct action at previous state.
- In forward state space search we used to need information about the successors of the current state now, for backward state-space search we will need information about the predecessors of the current state.
- Here the problem is that there can be many possible goal states which are equally acceptable. That is why this approach is not considered as a practical approach when there are large numbers of states which satisfy the goal.
- Let us see flight example, here you can see that the goal state is flight 1 is at location B and flight 2 is also at location B. We can see in Fig. 3.19.1. If this state is checked backwards we have two acceptable states in one state only flight 2 is at location B, but flight 1 is at location A and similarly in 2nd possible state flight 1 is already at location B, but flight 2 is at location A.
- As we search backwards from goal state to initial state, we have to deal with partial information about the state, since we do not yet know what actions will get us to goal. This method is complex because we have to achieve a conjunction of goals.

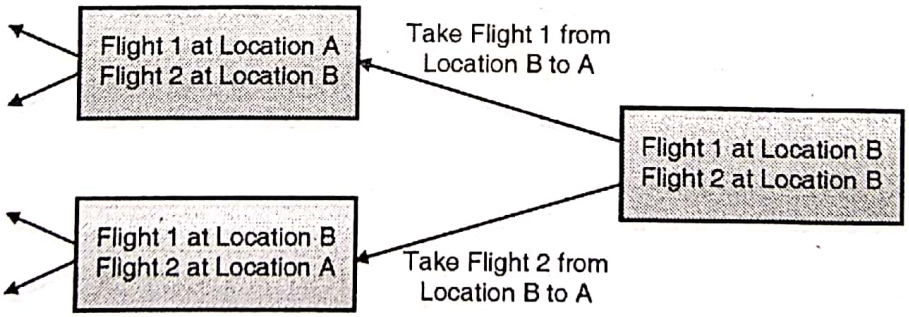


Fig. 3.19.1 : State-space graph of a regression planners

- In this Fig. 3.19.1, rectangles are goals that must be achieved and lines shows the corresponding actions.

Regression algorithm

1. Firstly predecessors should be determined :
   - To do this we need to find out which states will lead to the goal state after applying some actions on it.
   - We take conjunction of all such states and choose one action to achieve the goal state.
   - If we say that "X" action is relevant action for first conjunct then, only if pre-conditions are satisfied it works.
   - Previous state is checked to see if the sub-goals are achieved.

Scanned by CamScanner

2. Actions must be consistent it should not undo preferred literals. If there are positive effects of actions which appear in goal then they are deleted. Otherwise Each precondition literal of action is added, except it already appears.

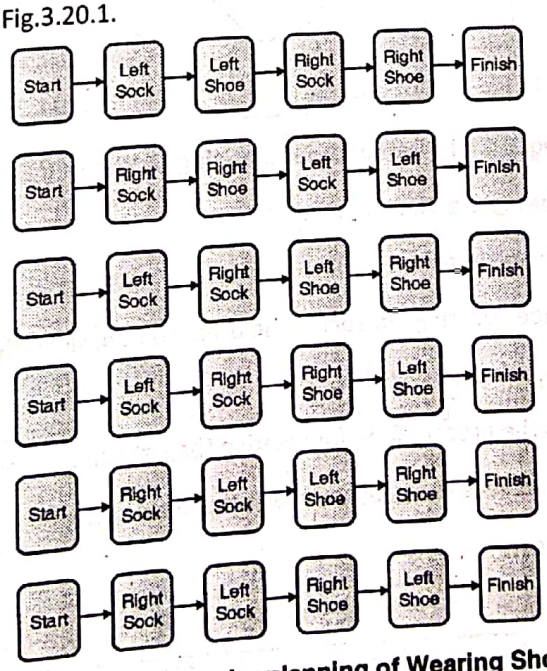
3. Main advantage of this method is only relevant actions are taken into consideration. Compared to forward search, backward search method has much lower branching factor.

3.19.1 Heuristics for State-Space Search

- Progression or Regression is not very efficient with complex problems. They need good heuristic to achieve better efficiency. Best solution is NP Hard (NP stands for Non-deterministic and Polynomial-time).
- There are two ways to make state space search efficient:
  - **Use linear method :**Add the steps which build on their immediate successors or predecessors.
  - **Use partial planning method :**As per the requirement at execution time ordering constraints are imposed on agent.

3.20 Total Order Planning (TOP)

- We have seen in above section that forward and regression planners impose a total ordering on actions at all stages of the planning process.
- In case of Total Order Planning (TOP), we have to follow sequence of actions for the entire task at once and to do this we can have multiple combinations of required actions. Here, we need to remember one most important thing which should be taken care of, is that TOP should take care of preconditions while creating the sequence of actions.
- For example, we cannot wear left shoe without wearing the left sock and we cannot wear the right shoe without wearing the right sock. So while creating the sequence of actions in total ordered planning, wearing left sock action should be executed before wearing the left shoe and wearing the right sock action should be executed before wearing the right shoe. As you can see in Fig.3.20.1.



Fig, 3.20.1 :Total order planning of Wearing Shoe

Scanned by CamScanner

- If there is a cycle of constraints then, total ordered planning cannot give good results. TOP can fail in non-cooperative environments. So we have Partial Ordered Planning method.

3.21 Partial Order Planning

MU – May 13, May 14, Dec. 14, May 15, Dec. 15

Q. Explain partial order planner with an example. (Dec. 15, 6 Marks)

Q. Define partial order planner. (May 13, Dec. 14, 5 Marks)

Q. Discuss partial order planning giving suitable example. (May 14, May 15, 10 Marks)

- In case of **Partial Ordered Planning (POP)**,ordering of the actions is partial. Also partial ordered planning does not specify which action will come first out of the two actions which are placed in plan.
- With partial ordered planning, problem can be decomposed, so it can work well in case the environment is non-cooperative.
- Take same example of wearing shoe to understand partial ordered planning.

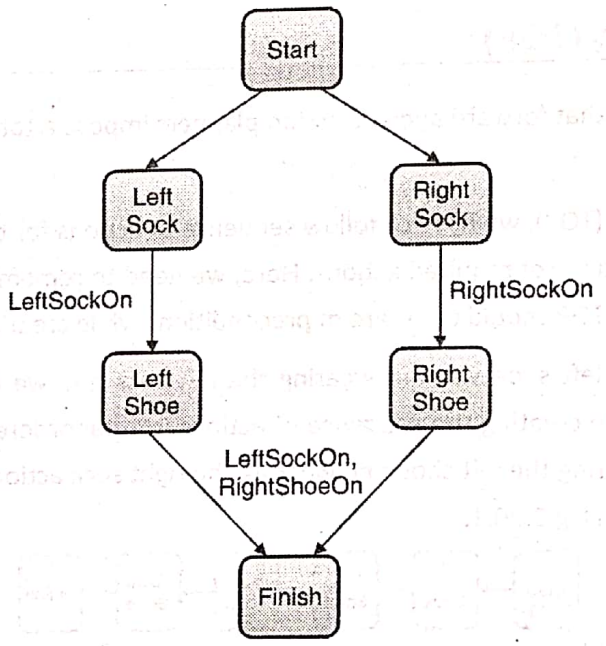


Fig. 3.21.1 : Partial order planning of Wearing Shoe

- **A partial order planning combines two action sequences**
  - First branch covers left-sock and left-shoe.
  - In this case to wear a left shoe, wearing left sock is the precondition, similarly.
  - Second branch covers right-sock and right-shoe.
  - Here, wearing a right sock is the precondition for wearing the right shoe.
- Once these actions are taken we achieve our goal and reach the finish state.

3.21.1 POP as a Search Problem

MU - Dec. 14, May 15, Dec. 15

Q. Define partial order planner. (Dec. 14, May 15, Dec. 15, 5/6 Marks)

Scanned by CamScanner

- If we considered POP as a search problem, then we say that states are small plans.
- States are generally unfinished actions. If we take an empty plan then, it will consist of only starting and finishing actions.
- Every plan has four main components, which can be given as follows:

1. **Set of actions**
   - These are the steps of a plan. Actions which can be performed in order to achieve goal are stored in set of actions component.
   - **For example :** Set of Actions={ Start, Rightsock, Rightshoe, Leftsock, Leftshoe, Finish}
   - Here, wearing left sock, wearing left shoe, wearing right sock, wearing right shoe are set of actions.
2. **Set of ordering constraints/ preconditions**
   - Preconditions are considered as ordering constraints. (i.e. without performing action "x" we cannot perform action "y")
   - **For example :** Set of Ordering ={Right-sock < Right-shoe; Left-sock < Left-shoe} that is In order to wear shoe, first we should wear a sock.
   - So the ordering constraint can beWear Left-sock < wear Left-shoe (Wearing Left-sock action should be taken before wearing Left-shoe)Or Wear right-sock < wear right-shoe(Wearing right-sock action should be taken before wearing right-shoe).
   - If constraints are cyclic then it represents inconsistency.
   - If we want to have a consistent plan then there should not be any cycle of preconditions.
3. **Set of causal links**
   - Action A achieves effect "E" for action B

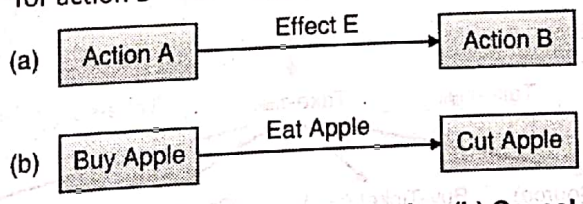


Fig.3.21.2 : (a) Causal Link Partial Order Planning (b) Causal Link Example

   - From Fig.3.21.2(b) you can understand that if you buy an apple it's effect can be eating an apple and the precondition of eating an apple is cutting apple.
   - There can be conflict if there is an action C that has an effect ¬E and, according to the ordering constraints it comes after action A and before action B.
   - Say we don't want to eat an apple instead of that we want to make a decorative apple swan. This action can be between A and B and It does not have effect "E".
     - **For example:** Set of Causal Links = {Right-sock->Right-sock-on → Right-shoe, Leftsock → Leftsockon → Leftshoe, Rightshoe → Rightshoeon → Finish, leftshoe → leftshoeon → Finish }.
     - To have a consistent plan there should not be any conflicts with the causal links.
4. **Set of open preconditions**
   - Preconditions are called open if it cannot be achieved by some actions in the plan. Least commitment strategy can be used by delaying the choice during search.
   - To have a consistent plan there should not be any open precondition

Scanned by CamScanner

3.21.2 Consistent Plan is a Solution for POP Problem

- As consistent plan does not have cycle of constraints; it does not have conflicts in the causal links and does not have open preconditions so it can provide a solution for POP problem.
- While solving POP problem operators can add links and steps from existing plans to open preconditions in order to fulfil the open preconditions and then steps can be ordered with respect to the other steps for removing the potential conflicts. If the open precondition is unattainable, then backtrack the steps and try solving the problem with POP.
- Partial ordered planning is a more efficient method, because with the help of POP we can progress from vague plan to complete and correct solution in a faster way. Also we can solve a huge state space plan in less number of steps, this is because search takes place only when sub-plans interact.

3.22 Hierarchical Planning

- Hierarchical planning is also called as plan decomposition. Generally plans are organized in a hierarchical format.
- Complex actions can be decomposed into more primitive actions and it can be denoted with the help of links between various states at different levels of the hierarchy. This is called as operator expansion.

For example :

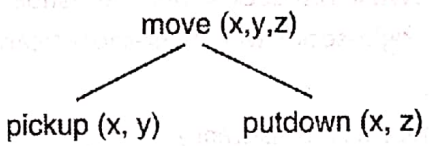


Fig. 3.22.1 : Operator expansion

- Fig. 3.22.2 shows, how to create a hierarchical plan to travel from some source to a destination. Also you can observe, at every level how we follow some sequence of actions.

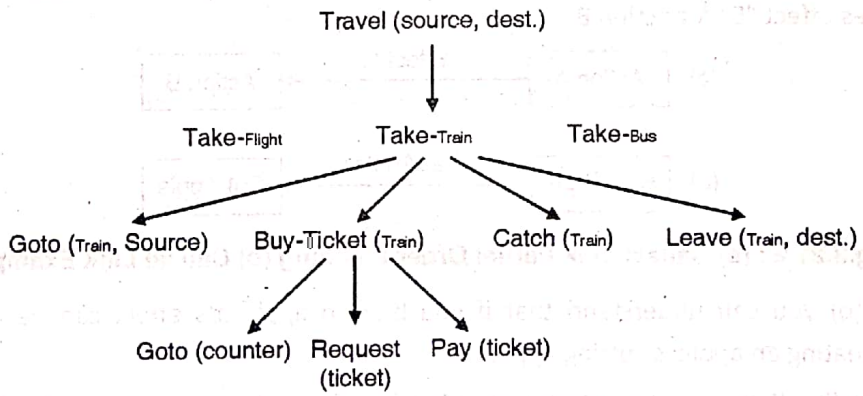


Fig. 3.22.2 : Hierarchical planning example

3.22.1 POP One Level Planner

- If you are planning to take a trip, then first you have to decide the location. To decide location we can search for various good locations from internet based on, weather conditions, travelling expenses, etc.
- Say we select Rajasthan, with one level planner, first we switch on PC, then we open browser, after that we open Indian Railways website booking site for ticket booking, then we enter the date, time, etc details to book the railway ticket. After that we will have to do hotel's ticket booking and so on.
- This type of planning is called **one level planning**. If the type of problem is simple then we can make use of one level planner. For complex problems, one level planner cannot provide good solution.

Scanned by CamScanner

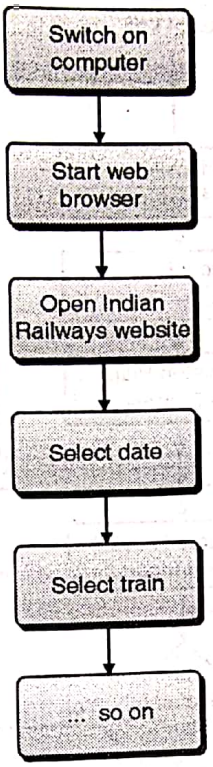


Fig. 3.22.3 : One Level planning example

3.22.2 Hierarchy of Actions

- In terms of major and minor actions, hierarchy of actions can be decided. Minor activities would cover more precise activities to accomplish the major activities. In case of above example, we can have railway Ticket Booking, Hotel Booking, Reaching Rajasthan, Staying and enjoying there, coming back are the major activities.
- While take a taxi to reach railway station, Have candle light dinner in palace, Take photos, etc are the Minor activities.
- In real world there can be complex problems.
- **For example :** A captain of a cricket team plans the order of 4 bowlers in 2 days of a test match(180 overs).Number of probabilities :$4^{180} = 16^{90}$.
- Motivation behind this planning is to reduce the size of search space. For plan ordering we have to try out a large number of possible plans. With plan hierarchies we have limited ways in which we can select and order primitive operators.
- In hierarchical planning major steps are given more importance. Once the major steps are decided we attempt to solve the minor detailed actions.
- It is possible that major steps of plan may run into difficulties at a minor step of plan. In such case we need to return to the major step again to produce appropriately ordered sequence to devise the plan.

3.22.3 Planner

1. First identify a hierarchy of major conditions.
2. Construct a plan in levels (Major steps then minor steps), so we postpone the details to next level.
3. Patch major levels as detail actions become visible.
4. Finally demonstrate.

Scanned by CamScanner

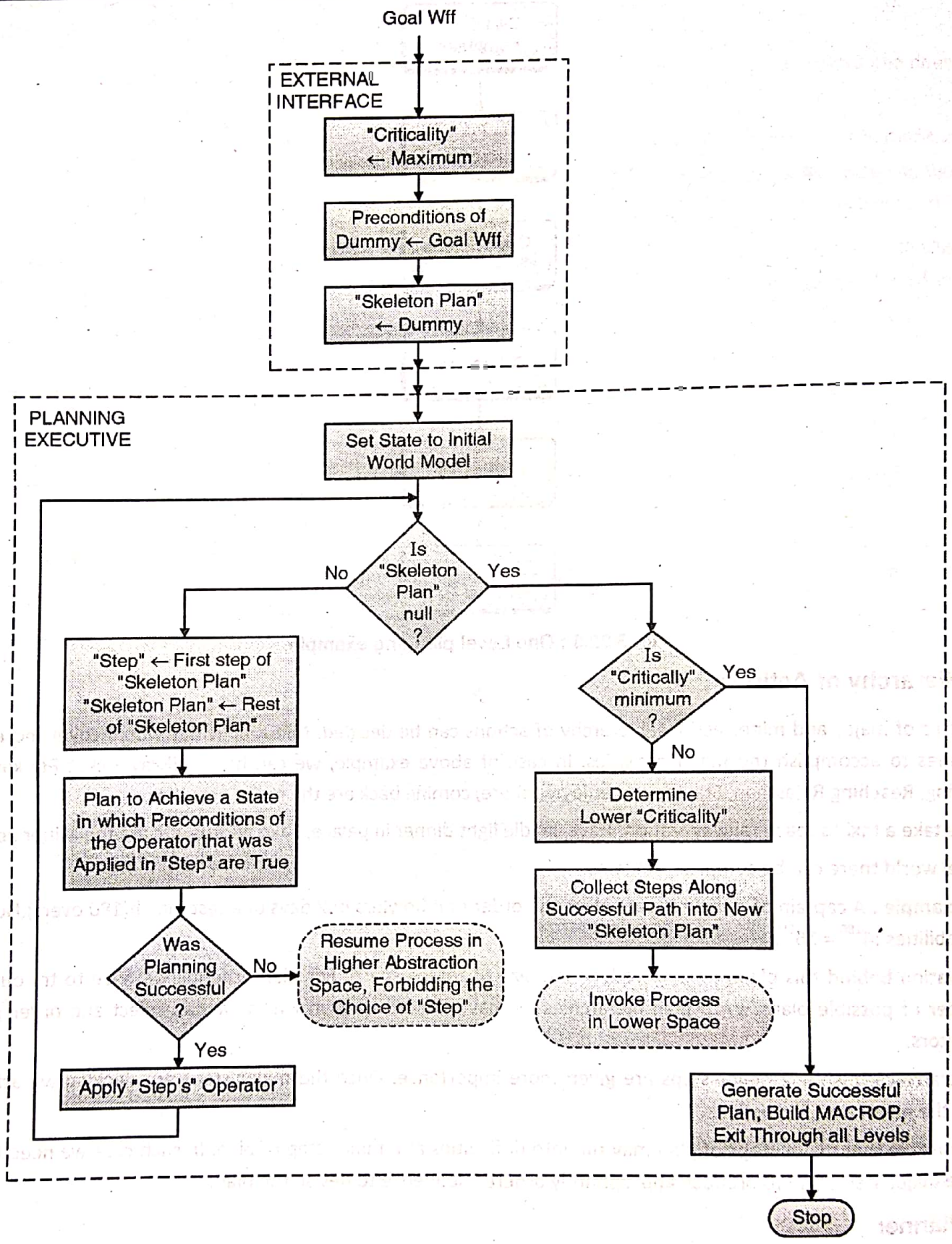


Fig. 3.22.4 : Planner

Example :

Actions required for "Travelling to Rajasthan" can be given as follows :

- Opening yatra.com (1)
- Finding train (2)
- Buy Ticket (3)

Scanned by CamScanner

- Get taxi(2)
- Reach railway station(3)
- Pay-driver(1)
- Check in(1)
- Boarding train(2)
- Reach Rajasthan (3)

1st level plan

Buy Ticket (3), Reach Railway Station(3), Reach Rajasthan (3)

2nd level plan

Finding train (2), Buy ticket (3), Get taxi(2), Reach railway station (3), Boarding train(2), Reach Rajasthan (3).

3rd level plan (final)

Opening yatra.com (1), Finding train (2), Buy ticket (3), Get taxi(2), Reach Railway station (3), Pay-driver(1), Check in(1), Boarding train(2), Reach Rajasthan (3).

3.23 Planning Languages

MU - May 13, Dec. 14

Q. Explain STRIPS representation of planning problem. **(May 13, Dec. 14, 5 Marks)**

- Language should be expressive enough to explain a wide variety of problems and restrictive enough to allow efficient algorithms to operate on it.
- Planning languages are known as action languages.

Stanford Research Institute Problem Solver (STRIPS)

- Richard Fikes and Nils Nilsson developed an automated planner called STRIPS (Stanford Research Institute Problem Solver) in 1971.
- Later on this name was given to a formal planning language. STRIPS is foundation for most of the languages in order to express automated planning problem instances in current use.

Action Description Language (ADL)

ADL is an advancement of STRIPS. Pednault proposed ADL in 1987.

Comparison between STRIPS and ADL

Sr. No.	STRIPS language	ADL
1.	Only allows positive literals in the states, **For example :** A valid sentence in STRIPS is expressed as ⇒ Intelligent^ Beautiful.	Can support both positive and negative literals. **For example :** Same sentence is expressed as ⇒ ¬Stupid ∧ ¬Ugly
2.	Makes use of closed-world assumption (i.e. Unmentioned literals are false)	Makes use of Open World Assumption (i.e. unmentioned literals are unknown)
3.	We only can find ground literals in goals. **For example :** Intelligent ∧ Beautiful.	We can find quantified variables in goals. **For example :** ∃x At (P1, x) ∧ At(P2, x) is the goal of having P1 and P2 in the same place in the example of the blocks

Scanned by CamScanner

Sr. No.	STRIPS language	ADL
4.	Goals are conjunctions **For example :** (Intelligent ∧ Beautiful).	Goals may involve conjunctions and disjunctions **For example :** (Intelligent ∧ (Beautiful ∨ Rich)).
5.	Effects are conjunctions	Conditional effects are allowed: when P:E means E is an effect only if P is satisfied
6.	Does not support equality.	Equality predicate (x = y) is built in.
7.	Does not have support for types	Supported for types **For example :** The variable p : Person

3.23.1 Example of Block World Puzzle

Fig. 3.23.1

Standard sequence of actions is

1. Grab Z and Pickup Z
2. Then Place Z on the table
3. Grab Y and Pickup Y
4. Then Stack Y on Z
5. Grab X and Pickup X
6. Stack X on Y

– Elementary problem is that framing problem in AI is concerned with the question of what piece of knowledge or information is pertinent to the situation.

– To solve this problem we have tom make an Elementary Assumption which is a Closed world assumption. (i.e. If something is not asserted in the knowledge base then it is assumed to be false, this is also called as "Negation by failure")

– Standard sequence of actions can be given as for the block world problem :

on(Y, table)	on(Z, table)
on(X, table)	on(Y, Z)
on(Z, X)	on(X, Y)
hand empty	hand empty
clear(Z)	clear(X)
clear(Y)	

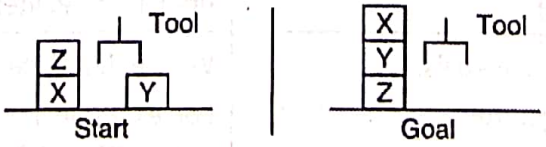


Fig. 3.23.2

Scanned by CamScanner

- We can write 4 main rules for the block world problem as follows :

	Rule	Precondition and Deletion List	Add List
Rule 1	pickup(X)	hand empty, on(X,table), clear(X)	holding(X)
Rule 2	putdown(X)	holding(X)	hand empty, on(X,table), clear(X)
Rule 3	stack(X,Y)	holding(X), clear(Y)	on(X,Y), clear(X)
Rule 4	unstack(X,Y)	on(X,Y), clear(X)	holding(X), clear(Y)

- Based on the above rules, plan for the block world problem:Start → goalcan be specified as follows:
  1. unstack(Z,X)
  2. putdown(Z)
  3. pickup(Y)
  4. stack(Y,Z)
  5. pickup(X)
  6. stack(X,Y)
- Execution of this plan can be done by making use of a data structure called "Triangular Table".

1	on (C, A) clear (C) handempty	unstuck (C, A)					
2		holding (C)	putdown (C)				
3	on (B, table)		hand empty	pickup (B)			
4			clear (C)	Holding (B)	stack (B,C)		
5	on (A, table)	clear (A)			Handem pty	pickup (A)	
6					clear (B)	holding (A)	stack (A, B)
7			on (C, table)		on (B, C)		on (A, B) clear (A)
	0	1	2	3	4	5	6

Fig. 3.23.3

- In a triangular table there are N + 1 rows and columns. It can be seen from the Fig. 3.23.4 that rows have 1 → n+1 condition and for columns 0 → n condition is followed. The first column of the triangular table indicates the starting state and the last row of the triangular table indicates the goal state.
- With the help of triangular table a tree is formed as shown below to achieve the goal state :

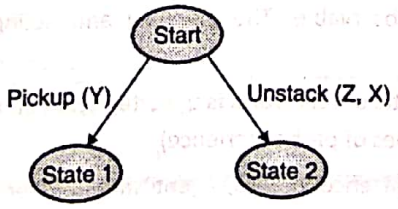


Fig. 3.23.4

- An agent (in this case robotic arm) can have some amount of fault tolerance. Fig. 3.23.5 shows one such example.

Scanned by CamScanner

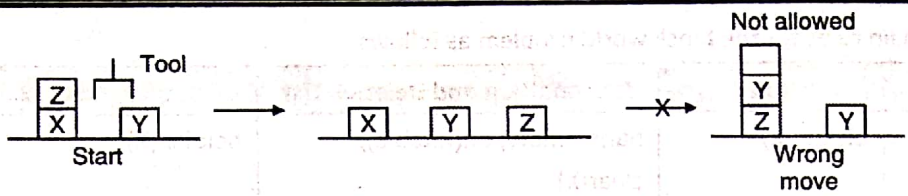


Fig. 3.23.5

3.23.2 Example of the Spare Tire Problem

- Consider the problem of changing a flat tire. More precisely, the goal is to have a good spare tire properly mounted onto the car's axle, where the initial state has a flat tire on the axle and a good spare tire in the trunk. To keep it simple, our version of the problem is a very abstract one, with no sticky lug nuts or other complications.
- There are just four actions: removing the spare from the trunk, removing the flat tire from the axle, putting the spare on the axle, and leaving the car unattended overnight. We assume that the car is in a particularly bad neighborhood, so that the effect of leaving it overnight is that the tires disappear.
- The ADL description of the problem is shown. Notice that it is purely propositional. It goes beyond STRIPS in that it uses a negated precondition, ¬At(Flat, Axle), for the PutOn(Spare, Axle) action. This could be avoided by using Clear (Axle) instead, as we will see in the next example.

Solution using STRIPS

- Init(At(Flat, Axle) ∧ At(Spare, Trunk))
- Goal(At(Spare, Axle)) Action(Remove(Spare, Trunk),
- PRECOND: At(Spare, Trunk)
- EFFECT: ¬ At(Spare, Trunk) ∧ At(Spare, Ground))
- Action(Remove(Flat, Axle),
- PRECOND: At(Flat, Axle)
- EFFECT: ¬ At(Flat, Axle) ∧ At(Flat, Ground))
- Action(PutOn(Spare, Axle),
- PRECOND: At(Spare, Ground) ∧ ¬ At(Flat, Axle)
- EFFECT: ¬ At(Spare, Ground) ∧ At(Spare, Axle))
- Action(LeaveOvernight)
- PRECOND
- EFFECT: ¬ At(Spare, Ground) ∧ ¬ At(Spare, Axle) ∧ ¬ At(Spare, Trunk) ∧ ¬ At(Flat, Ground) ∧ ¬ At(Flat, Axle))

3.24 Planning and Acting in the Real World

- In earlier sections, we have discussed planning that is observable and deterministic. Real world problems however are not predictable and completely unobservable. The planning and acting on real world problems require more sophisticated approach.
- One of the most important characteristic of real world is uncertainty. In an uncertain environment, it is very important for agent to rely upon its percepts (series of past experience)
- Whenever some unexpected condition encountered, agent should refer its percept and should change course of action accordingly. In other words agent should be able to cancel or replace the currently executing plan with some other more suitable and reliable plan if something unexpected happens.

Scanned by CamScanner

- It should be noted that real world itself is not uncertain but human perception related to world is uncertain. In artificial intelligence, we try to give human perception ability to machine, and hence machine also receives the perception of uncertainty about the real world. So machine has to deal with incomplete and incorrect information like human does.
- Determining the condition of state depends on available knowledge. In real world, knowledge availability is always limited, so most of the time, conditions are non deterministic.
- The amount or degree of indeterminacy depends upon the knowledge available. The inter determinacy is called "bounded indeterminacy" when actions can have unpredictable effects.
- Four planning strategies are there for handling indeterminacy :

(i) Sensorless planning
(ii) Conditional planning
(iii) Execution monitoring and replanning
(iv) Continuous planning

(i) Sensorless planning

Sensorless planning is also known as conformant planning. These kinds of planning are not based on any perception. The algorithm ensures that plan should reach its goal at any cost.

(ii) Conditional planning

Conditional plannings are sometimes termed as contingency planning and deals with bounded indeterminacy discussed earlier. Agent makes a plan, evaluate the plan and then execute it fully or partly depending on the condition.

(iii) Execution monitoring and replanning

In this kind of planning agent can employ any of the strategy of planning discussed earlier. Additionally it observes the plan execution and if needed, replan it and again executes and observes.

(iv) Continuous planning

Continuous planning does not stop after performing action. It persist over time and keeps on planning on some predefined events. These events include any type unexpected circumstance in environment.

3.25 Multi-Agent Planning

- Whatever planning we have discussed so far, belongs to single user environment. Agent acts alone in a single user environment.
- When the environment consists of multiple agent to, then the way a single agent plan its action get changed.
- We have a glimpse of environment where multiple agents have to take actions based on current state. The environment could be co-operative or competitive. In both the cases agent's action influences each other.
- Few of the multi agent planning strategies are listed below :

(i)	Co-operation	(ii)	Multi body planning
(iii)	Co-ordination mechanisms	(iv)	Competition

Scanned by CamScanner

(i) Co-operation

In co-operation strategy agents have joint goals and plans. Goals can be divided into sub goals but ultimately combined to achieve ultimate goal.

(ii) Multibody planning

Multi body planning is the strategy of implementing correct joint plan.

(iii) Co-ordination mechanisms

These strategies specify the co-ordination between co-operating agents. Co-ordination mechanism is used in several co-operating plannings.

(iv) Competition

Competition strategies are used when agents are not co-operating but competing with each other. Every agent wants to achieve the goal first.

3.26 Conditional Planning

- Conditional planning has to work regardless of the outcome of an action.
- Conditional Planning can take place in Fully Observable Environments (FOE) where the current state of the agent is known environment is fully observable. The outcome of actions cannot be determined so the environment is said to be nondeterministic.

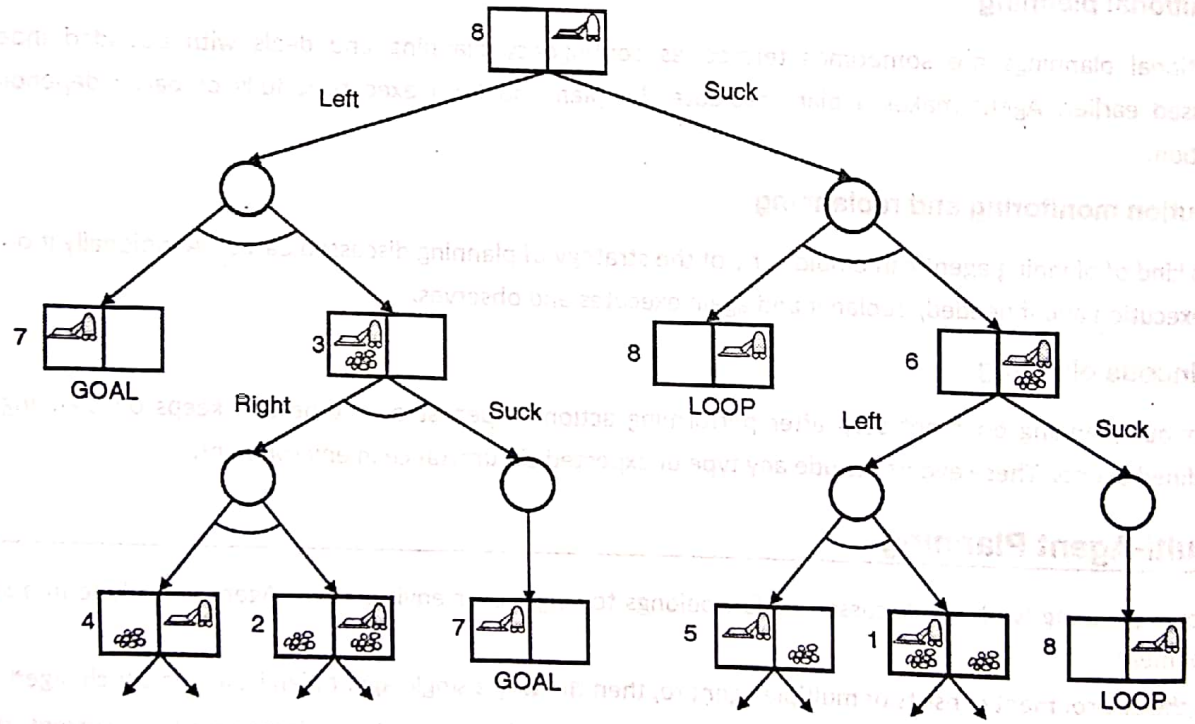


Fig. 3.26.1 : Conditional planning - vacuum world example

- In conditional planning we can check what is happening in the environment at predetermined points of the plan to deal with ambiguous actions.
- It can be observed from vacuum world example ,Conditional Planning needs to take some actions at every state and must be able to handle every outcome for the action it takes. A State node is represented with a square and chance node is represented with a circles.
- For a state node we have an option of choosing some actions. For a chance node agent has to handle every outcome.
- Conditional Planning can also take place in the Partially Observable Environments (POE) where, we cannot keep a track on every state. Actions can be Uncertain because of the Imperfect sensors.

Scanned by CamScanner

- In vacuum agent example if the dirt is at Right and agent knows about Right, but not about Left. Then, in such cases Dirt might be left behind when the agent, leaves a clean square. Initial state is also called as a state set or a belief state.
- Sensors play important role in Conditional Planning for partially observable environments. Automatic sensing can be useful; with automatic sensing an agent gets all the available percepts at every step. Another method is Active sensing, with which percepts are obtained only by executing specific sensory actions.

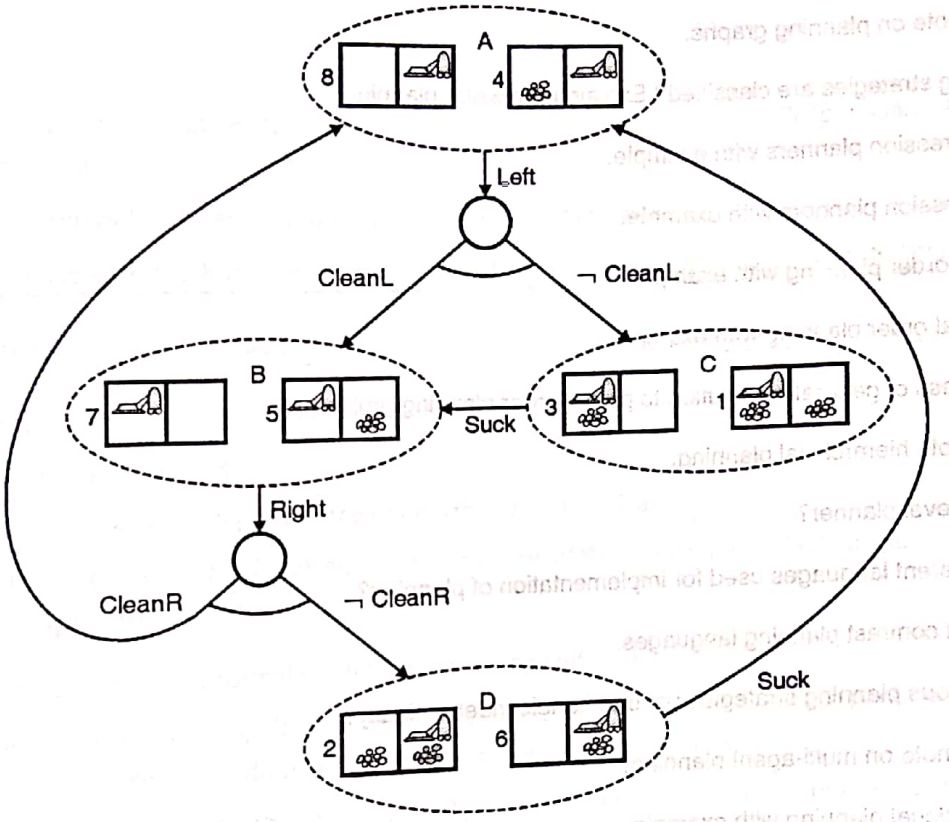


Fig. 3.26.2 :Conditional Planning - vacuum world example (condition 2)

Review Questions

Q. 1 What is Knowledge Based Agent?

Q. 2 Describe WUMPUS WORLD Environment. Specify PEAS properties and type of environment for the same.

Q. 3 What is Logic? Explain various knowledge representation techniques.

Q. 4 What is propositional logic? Write syntax and semantics and example sentences for propositional logic.

Q. 5 Explain the inference process in case of propositional logic with suitable examples.

Q. 6 Explain Horn Clause with example.

Q. 7 What is first order logic? Write syntax and semantics of FOL with example.

Q. 8 What is unification and lifting?

Q. 9 Explain inference process in FOL using Forward Chaining.and Backward Chaining.

Q. 10 Write a short note on : Resolution.

Scanned by CamScanner

Q. 11 What is planning in AI?

Q. 12 Explain planning problem.

Q. 13 Explain goal of planning with supermarket example.

Q. 14 What are the major approaches of planning? Explain conditional planning with example.

Q. 15 Write short note on planning graphs.

Q. 16 How planning strategies are classified? Explain regression planning.

Q. 17 Explain progression planners with example.

Q. 18 Explain regression planners with example.

Q. 19 Explain total order planning with example.

Q. 20 Explain partial order planning with example.

Q. 21 Explain process of generating solution to partial order planning problem.

Q. 22 Write short note hierarchical planning.

Q. 23 What is one level planner?

Q. 24 What are different languages used for implementation of planning?

Q. 25 Compare and contrast planning languages.

Q. 26 What are various planning strategies used to handle indeterminacy?

Q. 27 Write a short note on multi-agent planning.

Q. 28 Explain conditional planning with example.

Scanned by CamScanner