CHAPTER 9

Planning

Learning Objectives

- To understand the concept and importance of planning
- To understand the role of planning in intelligent systems
- To understand the importance of planning in autonomous agents
- To study and understand the application and usage of planning in AI
- To acquire knowledge about the algorithms and the methods in planning
- To understand the planning systems and its impact on uncertainty and multi-agent systems

INTRODUCTION

Planning is required to convert objectives into actions. To perform any activity effectively, one needs to plan. Every morning one gets ups and decides the agenda and plans for that day. A mother has her own set of do's and for that she plans according to the timings of her kids coming from the school. Based on different constraints related to timings of other family members, she plans all her household activities including shopping, laundry visit, etc. Similarly, students plan their study based on the exam schedule and other constraints. They take into account examination schedule, time available at hand, subjects he/she is good at and a number of other factors related to this activity. Planning for the studies includes scheduling of all the subjects in such a way so that a particular subject is studied in a specified time; thus, it deals with scheduling and ordering of studies! There are dependencies like subject A is mandatory to understand subject B, so subject A should be studied prior to subject B.

You have a lot of constraints too while planning—number of days left for the exam, subject category (simple/hard), number of chapters to study, notes availability and so on. Still, you have to plan to achieve some goal. The goal could be to get distinction or just to clear all the subjects. So, planning involves the steps to be taken to reach the goal optimally. Thus, we can say that it determines how and when to do the necessary steps, given the set of goals.

Further, in the real world, without prior knowledge, one ends up in searching exhaustively for the goal state. Let us take an example of problem-solving agent. An exhaustive search is required to reach the goal. It would typically be a search that may have to traverse back and re-apply (re-think) of the actions. How nice it would be if we were to have some heuristic before taking any decisions (in real life just not possible)! If we could have the problem divided into different tasks, it would make the things simpler. Having some guidelines as to how to travel to the goal state with some intermediate states known would make it much better. Considering all these situations, there is a need to have planning to solve the problem, optimally.

What is the role of planning in case of AI? Intelligent systems need to be aware of what is to be done and how it can be done. Planning allows exploring different possibilities; it makes intelligent systems more flexible, and hence, increases their decision-making capabilities. In order to plan, the problem needs to be well-understood. Planning needs proper representation of what needs to be achieved. It involves the reasoning as well as the techniques that are applied. Planning plays a major role in various tasks like spacecrafts missions, robotics, specific military tasks and so on. Let us start a detailed study on the concepts of planning.

9.1 PLANNING PROBLEM

So, what is the basic planning problem? The *planning problem* is actually the question that how to go to the next state or the goal state from the current state. As discussed earlier, it involves two things 'how' and 'when'. It definitely involves selecting appropriate action and sequencing them so that the required action is invoked first. Let us make things more clear and more technical. The planning problem is defined with:

1. Domain model
2. Initial state
3. Goal state (next state)

The domain model defines the actions along with the objects. It is necessary to specify the

operators too that actually describe the action. Along with this, information about actions and state constraints while acting should also be given. This entirely formulates the domain model. The *initial state* is the state where any action is yet to take place (the stage when the exam schedule is put up!). The *final state* or the *goal state* is the state which the plan is intended to achieve. It constitutes of the facts that will be true once your plan gets completed (getting an all clear in the subjects!).

So, we work on a strategy that actually forms the plan. The decision of the sequence in which the actions are performed too is a plan.

9.1.1 Components of Planning System

Let us refer to the same example of the planning for the exam to understand the components of planning system. A student who wants to get an all clear (i.e., to score marks above cut-off in all subjects) applies some selection criteria to begin with the study of any subject. He recognises the difficulties that he is likely to face. What he is actually doing is forming the components of the planning system. We highlight them in technical terms for planning in AI. The components are as follows:

1. Selection of the best rule/action by the use of heuristic (say selection of a simple subject—this would be based on the past experience or inputs from the seniors or lectures attended.)
2. Taking action and applying the rule (studying of the subject, so as to proceed towards the next stage)
3. Understanding and recognising when the goal is reached or the solution is found (a position is reached where he can score more than cut-off marks.)
4. Understanding and recognising the dead ends (got stuck with some difficult topic which takes him nowhere.)
5. Understanding and recognising partial solution that is correct and making efforts to make it entirely accurate. (In position to get more than cut-off marks in all subjects except mathematics and then strategise to reach the goal state.)

9.1.2 Basic Planning Representation

To plan, one should be able to represent the problem properly. Representation of the planning problem is mapping of the states, actions and the goals. For the representation, the language used should be concrete, understandable and expressive. In a broader perspective, there are different representation methods or ways that are followed like propositional, first order, state variable. Figure 9.1 depicts the high-level diagram for planning.

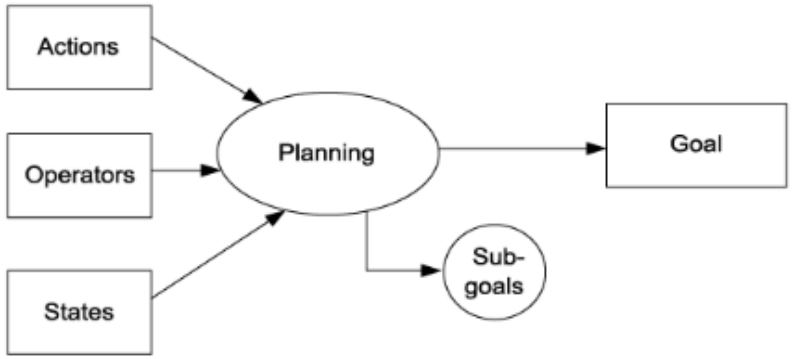

Figure 9.1 Planning.

Planning essentially needs the representation in terms of time. This is required so that we are able to reason regarding the actions that are to be taken along with the reactions that we get back. Let us proceed towards the representations of the states, goals and the actions.

State Representation

States are the representation of the facts. States are represented as conjunction. It comprises the positive literals that specify the state. We are already familiar with the first order logic and propositional logic. Any of them can be used for representation. In the state representation, while using the first order logic, the representation has to be easily understood. There has to be grounding (variables are replaced by constants) also in the representation. For example, kind(Richa) ^ hardworking(Richa) can be a state unless some actions are performed or friends(Richa, Hema) ^ friends(Seema, Meena). One more example can be On(Apple, Table) that represents some state. But we cannot have friends (x, y), as said earlier about grounding. Further, the representation also assumes that the conditions we do not specify explicitly are not accounted or rather are not considered to be true. This is also referred to as *closed world assumption*. So, the objects that are specified in the states only exist in the world. In case of propositional literals, we can have a state as kind ^ hardworking. (STRIPS representation follows the first order literals, the details of STRIPS will be discussed later.)

Goal Representation

Goal is most often a partially specified state. A state or say proposition is said to achieve or satisfy the given goal if it consists of all the objects required for the goal or may be some other too. As an example, if the goal is kind^hardworking, then a state that has kind^hardworking^pretty fulfils the goal.

Action Representation

Whenever we decide to do something, we are aware of the state we are in and what possibly the effects are. When it comes to the mapping of the actions for an agent or in case of robots, the pre and the post situations need to be specified. These can also be called as *pre-conditions* and the after effects are called *post-conditions*. For example, an action to drive from one place to another can be mapped as follows:

Action(drive(c,from,to))
Pre-condition: at(c,from)^car(c)
Post-condition: ~at(c,from)^at(c,to)

Here, in the post-condition, where we have written ~at(c,from) indicates that the state is deleted or is to be removed. In some cases, add and delete list can also be used.

In case of a state variable representation, the state comprises different state variables. The action here is defined as a partial function on the states.

So, we have discussed about the representation part, but how can the actions be applied actually? It follows the following criteria:

1. A substitution is to be identified for the variables. That is, for the current state that exists, identify the action with a pre-condition that satisfies the current state (the current state can be subset of the pre-condition).
2. Apply this substitution (for whatever part of the current state it is applicable).
3. Add the post-condition (effects) to the remaining subset of the current state if any.

9.2 SIMPLE PLANNING AGENT

We have been familiar with the agent world in the previous chapters. One would have easily guessed that the *planning agent* is the one that plans its actions. We have detailed the representation in the previous section and it helps us in understanding the planning agent and how it acts upon the environment. A planning agent is the one that makes use of knowledge and problem solving to get the goal. It performs the following steps:

1. Defining a goal (a goal is initially set by the agent).
2. Planning (a plan is built).
3. Taking action (actions are invoked as per the plan).

The process continues each time with some new goal. Things are simple to understand in the planning agent. (It is the same way we decide when we have some goal set). But since it is an agent environment, there have to be some assumptions. (Recollect the environment types an agent has). So, what does an agent assume?

1. The actions occur one at a time (no parallel execution) and that they cannot be further divided.
2. There is no uncertainty and the agent is well-versed with the outcome that is specified with the action.
3. Rules/things which are not specified are assumed to be false.
4. The agent's action causes the changes in the state and it has the knowledge about it. So, the environment is fully observable.

9.3 PLANNING LANGUAGES

After briefing about the representation, let us have a look at the different planning languages. We know that a language is the one that should be expressive. Again referring to the earlier section of representation, is this a part of the language? The answer is yes. Every planning language makes use of the representation schema so that the algorithms can be operated on it. We highlight three languages here.

1. Stanford Research Institute Problem Solver (STRIPS)
2. Action Description Language (ADL)
3. Planning Domain Description Language (PDDL)

9.3.1 Stanford Research Institute Problem Solver (STRIPS)

STRIPS is a language that is historically important. It was developed in 1970's at Stanford for the first intelligent robot. It makes use of the first order predicates. Hence, the representation structure that we have discussed earlier with the first order logic essentially belongs to the STRIPS. STRIPS allows function-free literals. We discuss one example for better understanding of STRIPS. Since the planning discussed is related to the agent's environment, we have the typical example of a robot. The example involves a robot, a cup of tea, guest and two rooms. We want the robot to get the tea and give it to the guest. The planning with STRIPS is done as follows:

Let us begin with the STRIPS representation for this example. Initial and final states are depicted in Figure 9.2.

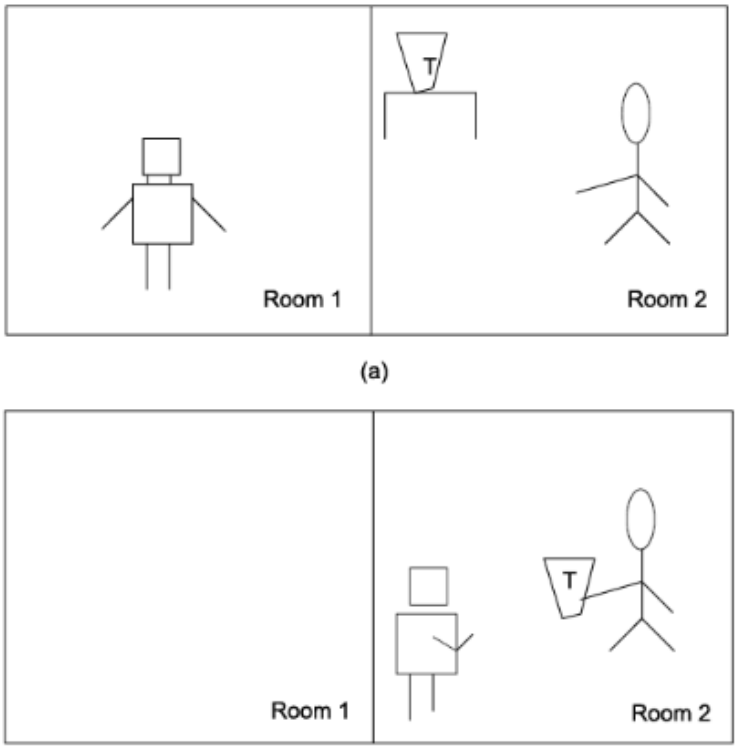

Figure 9.2 (a) Initial stage, (b) Goal state.

State Representation

The possible states can be as follows:

1. in(robot,room1)^in(tea,room2)^in(guest,room2)
2. in(robot,room1)^in(tea,room1)^in(guest,room1)
3. in(robot,room2)^in(tea,room1)^in(guest,room1) and so on.

Let the goal state be in(guest,room2)^in(robot,room2)^in(tea,room2)^have(tea,guest).Let us assume the first case in the possible states to be the initial state.

Action Representation

The operators or the actions have name, pre and post-conditions. (Post-conditions can also be represented as add-list or delete-list.)

Let us take an action.

Action 1: move-to-room2

Pre-condition: in(robot,room1)
Post-condition: add-list: in(robot,room2)
delete-list: in(robot,room1)

There can be other actions like pick-tea, where the robot is in room2 and the tea as well, and the post-condition is hold(tea,robot). In such action, the delete-list can be empty, which is valid. Let us map it.

Pick-tea

Pre-condition: in(robot,room2)^in(tea,room2)
Post-condition: add-list: hold(tea,robot)
delete-list: NULL{empty-list}

In this manner, STRIPS makes use of lists, actions and operates.

Semantics

Whenever the pre-condition is false, the action cannot occur, as the state does not exist and it cannot be applied. But remember the order in which the add and delete lists operate is important.

From this, we can conclude that the STRIPS language is easy to understand. The use of propositional logic makes the implementation and building of the knowledge base easy. Due to this, inference and reasoning become simple. But the language is restricted. It assumes propositions in limited numbers. This can be a major drawback when considered with a different domain, where there could be more involvement of large number of propositions.

9.3.2 Action Description Language (ADL)

STRIPS language lacks the expressive power. It can be extended very well to overcome some of the limitations and ADL does that. The properties of ADL are as follows:

1. It allows negative literals.
2. It makes use of quantified variables along with the disjunctions and the conjunctions.
3. Conditional post-conditions are allowed.
4. Variables with different types at the same time are allowed and also equality property is available.

For example, consider the car driving case, with STRIPS, it will be

Action(drive(c,from,to))
Pre-condition: at(c,from) ^ car(c)
Post-condition: ~at(c,from) ^ at(c,to)

(Remember in STRIPS, the post-condition means remove the 1st 'at' condition and add the 2nd 'at' condition.)

With ADL, the same action is represented as follows:

Action(drive(c,from,to))
Pre-condition: at(c,from) ^ car(c) ^(from≠to)
Post-condition: ~at(c,from) ^ at(c,to)

ADL, thus, comes out to be more expressive language as compared to STRIPS and is looked upon for building reasoning approaches.

9.3.3 Planning Domain Description Language (PDDL)

It is a standardisation of the planning languages. It is defined by the researchers as a standard language. We can say it is a superset of STRIPS and ADL that allows features like

1. Objects can have type specifications.
2. It can have negative pre-conditions.
3. The add and delete lists can be conditional.
4. In some cases, it also allows numeric values.

PDDL is used in case of classical planning tasks. The planning tasks in PDDL are separated into domain file and problem file. A domain file consists of

(define (domain <Domain_name>)
<PDDL code for the predicates>
<PDDL code for the actions>
....)

A problem file comprises

(define (problem <problem_name>)
(:domain <Domain_name>)
<PDDL code for the objects>
<PDDL code for the initial state>
<PDDL code for the goal specification>
)

A domain_name identifies the planning domain, whereas a problem_name is the one that identifies the planning task. Most of the planners require this type of specification of PDDL.

9.4 BLOCKS WORLD

The blocks world is a known example that is used to demonstrate the planning using STRIPS. It is simple to understand and most important well behaved.

What is the blocks world? It consists of

1. A table or say a table top
2. Identical blocks that have unique letters on them.
3. The blocks can be put one on other to form a stack (a tower that has no height restriction).
4. This stack is built with a robot arm. The arm can perform operations of lifting a single block at a time and placing it. Operators and states are used, where logic is applied for the arm.

Let us say we have the following initial and goal states. This is a very simple example with just two blocks (Figure 9.3).

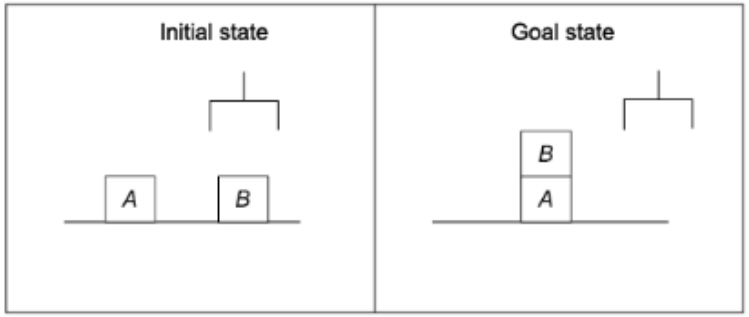

Figure 9.3 Blocks worked—Initial and goal stages.

The predicates used here are given below:

On(*A*,table): Block *A* is on the table.
On(*B*,table): Block *B* is on the table.
On(*B*,*A*): Block *B* is on block *A*.
Clear(*A*): Block *A* has nothing on it.
Clear(*B*): Block *B* has nothing on it.
Holding(*B*): Robot hand is holding *B*.
Empty-hand: The hand is not holding anything.

The state representation can be as follows:

On(*A*,table)^On(*B*,table)^Clear(*A*)^Clear(*B*)^Empty-hand

On(*A*,table)^Holding(*B*)^Clear(*A*)
On(*A*,table)^On(*B*,*A*)^Clear(*B*)^Empty-hand

The initial state is:

On(*A*,table)^On(*B*,table)^Clear(*A*)^Clear(*B*)^Empty-hand

The goal state is:

On(*A*,table)^On(*B*,*A*)^Clear(*B*)^Empty-hand

The actions that can take place are given below:

1. Unstack(*B*,*A*): To lift block *B* from *A*
   Pre-condition: Empty-hand^On(*B*,*A*)^Clear(*B*)^On(*A*,table)
   Post-condition: Delete-list: Empty-hand^On(*B*,*A*)^Clear(*B*)
   Add-list: Holding(*B*)^Clear(*A*)^On(*A*,table)
2. Stack(*B*,*A*): to place block *B* on *A*
   Pre-condition: Holding(*B*)^Clear(*A*)^On(*A*,table)
   Post-condition: Delete-list: Holding(*B*)^Clear(*A*)
   Add-list: On(*B*,*A*)^Clear(*B*)^Empty-hand^On(*A*,table)
3. Lift(*B*): To lift the block *B* from the table
   Pre-condition: On(*B*,table)^Clear(*B*)^Empty-hand^On(*A*,table)^Clear(*A*)
   Post-condition: Delete-list: On(*B*,table)^Clear(*B*)^Empty-hand
   Add-list: Holding(*B*)^On(*A*,table)^Clear(*A*)
4. Place(*B*): To put the block *B* on table
   Pre-condition: Holding(*B*)^On(*A*,table)^Clear(*A*)
   Post-condition: Delete-list: Holding(*B*)
   Add-list: On(*B*,table)^Clear(*B*)^Empty-hand^On(*A*,table)^Clear(*A*)

All the actions discussed here may not be applicable to the two blocks problem, but required when the number of blocks are more. The lift, place, stack and unstack can be with different quantities of blocks as per the problem definition.

9.5 GOAL STACK PLANNING

After studying the blocks world, we introduce the goal stack planning. This approach is used in STRIPS and is one of the very early approaches. In this approach, goal stacks are maintained to carry out the task. With the blocks world example, let us consider the initial and goal states given in Figure 9.4.

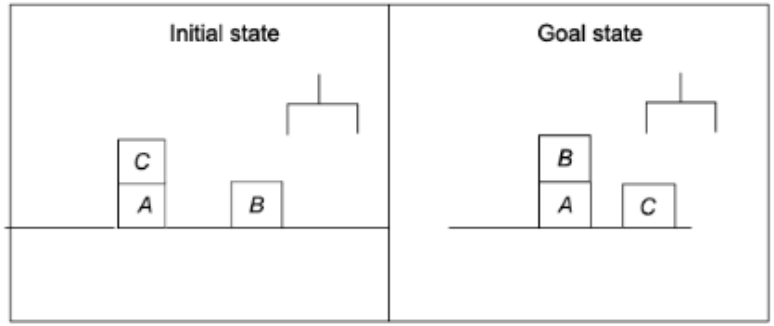

Figure 9.4 Goal stack planning—Initial and goal states.

In goal stack planning, a stack of goals is maintained. Accordingly, the corresponding actions are carried out to get the solution.

The initial state here is

On(*A*,table)^On(*B*,table)^On(*C*,*A*)^Clear(*C*)^Clear(*B*)^Empty-hand

The goal state is

On(*A*,table)^On(*B*,*A*)^On(*C*,table)^Clear(*C*)^Empty-hand^Clear(*B*)

For the goal, the following steps are carried (For simplicity, let us neglect the clear and empty-hand conditions here, as things will get more complicated.):

The refined goal will be

On(*A*,table)^On(*C*,table)^On(*B*,*A*)

The stack for the goal will be

Stack 1
On(*C*,table)
On(*B*,*A*)
On(*B*,*A*)^On(*C*,table)^On(*A*,table)

Here, the last operation leads to final goal, where *A* is already placed on the table.
So, for the On(*C*,table), we need

Place(*C*) and Unstack(*C*,*A*)

So, we replace the On(*C*, table) and add this to it.

Unstack(*C*,*A*)
Place(*C*)
On(*B*,*A*)

On(*B*,*A*)^On(*C*,table)^On(*A*,table)

For unstack, the pre-conditions are to be added. This is done in the following way:

On(*A*,table)
Clear(*C*)
On(*C*,*A*)
Empty-hand
Empty-hand^On(*C*,*A*)^Clear(*C*)^On(*A*,table)
Unstack(*C*,*A*)
Place(*C*)
On(*B*,*A*)
On(*B*,*A*)^On(*C*,table)^On(*A*,table)

Since it is observed that the first state of *A* block on the table is true, hence we remove it. Further, Clear *C* is also valid, and hence, it is also removed. So, these exist for the other two conditions of On(*C*,*A*) and Empty-hand. Since all the pre-conditions are valid, so we can proceed. They are removed to move further. Finally, the stack has:

Unstack(C,A)
Place (*C*)
On(*B*,*A*)
On(*B*,*A*)^On(*C*,table)^On(*A*,table)

Since all pre-conditions are satisfied for the unstack operation to occur, we record this action to be the first step that is required to be carried out in the planning sequence. For Place (*C*), the pre-conditions are—the hand should be holding *C* and then only it can be placed. So, the pre-conditions are added, and since all are valid, this action is also finalised.

Unstack(C,A)
Place(C)
On(*B*,*A*)
On(*B*,*A*)^On(*C*,table)^On(*A*,table)

In this way, the sequence is continued. There could be a case, where multiple options can exist. So, there could be two or more goal stacks, where the operation has to be carried out. Once the first goal is achieved, the next goal is looked upon. Similarly, the actions and the conditions are added.

Now, consider a case, where the second goal is itself contained in the first goal, say if the order is changed, for *B* to be on *A*, the goal of *C* to be on the table becomes contained in it. So, naturally, the sub-goal is reported to be reached.

Thus, we have discussed the overview of the goal stack planning. Now, the query arises here—is this approach an efficient one? For smaller problems, it is fine, but with larger ones, it becomes complicated. The case could occur, where the action that is not required is carried out, and further, it adds to the complications. There can be options to go back and recall the step that has been done in order to undo it. Still, there can be more and better methods to design and extract a plan instead of this one.

9.6 MEANS ENDS ANALYSIS

Means-ends Analysis is one of the important concepts that is used for problem solving and planning. The notion of means-ends analysis is same as what is used in STRIPS. Means-ends analysis is dependent on a set of rules. The rules are used for the transformation from one state to another. The pre-conditions and the post-conditions are included in it. A *difference table* is used to represent the difference between the current state and the next state. The action selection is based on this difference. It also follows a backward search to locate an action that has the required pre-conditions.

The steps carried out in means-ends analysis are given below:

1. Computing the difference between the current/initial state and the goal/next state.
2. Based on the satisfaction of preconditions, recommending the action (This reduces the difference between them).
3. Checking the possibility of executing the action (If not possible, then this current state is treated as the goal state and recursive call takes place).

Let us take an example. Suppose you are new in a city and you want to purchase a laptop. You do not know where the shop is. You have a contact number of your friend, who stays in that city and can help you with the address. With means-ends analysis, the difference table can be (only some rules are shown):

1. If you want to purchase a laptop then recommended actions are visit_shop or purchase online.
2. If you do not know the address then recommended actions are call friend or use tools like Google maps to get it, and so on.

For each of the recommended action, there is a pre and post-condition.

For example, for the action visit_shop, pre-condition can be—you should know the location to the shop, and the post-condition can be—you have the laptop purchased.

This set is formed for all the actions in the difference table.

Now, assume that you are at the first state, where you want to purchase the laptop. Since the pre-condition of the location of the shop to have the rule1 executed is not satisfied, the recursive call takes place with the condition of location not known and so on.

9.7 PLANNING AS A STATE-SPACE SEARCH

There are two approaches to planning: state-space planning and plan-space planning. What we intend to do in both the cases is to find a plan.

A *state-space planning* is the one that works at the states and the operators. This is also called *planning as a state-space search*. In this case, the plan is found as a search through the search space. Here, the search takes place in both the directions—forward and backward. In case of *plan-space planning*, the search is carried out through the space of plans. So, naturally, we need to begin with some plan that might be incorrect or incomplete. The ordering in the actions is done so as to get the accurate plan. It involves additions or deletions of the steps as well.

Let us discuss the first case, i.e., planning with state-space search in detail.

As discussed earlier, the planning can be from the initial state to the goal state or in reverse way. It is called *forward state-space search* and *backward state-space search*. A forward state-space search is called *progression planning*, whereas backward state-space search is called *regression planning*.

Progression Planning

This planning starts with the initial state and proceeds with search. It follows the effects of the possible actions. So, the sequence of actions is considered till we are able build or form a sequence to reach the goal.

Figure 9.5 represents the forward search for a case in which a robot (say) is in room1, and the tea and guest are in room2 and room3.

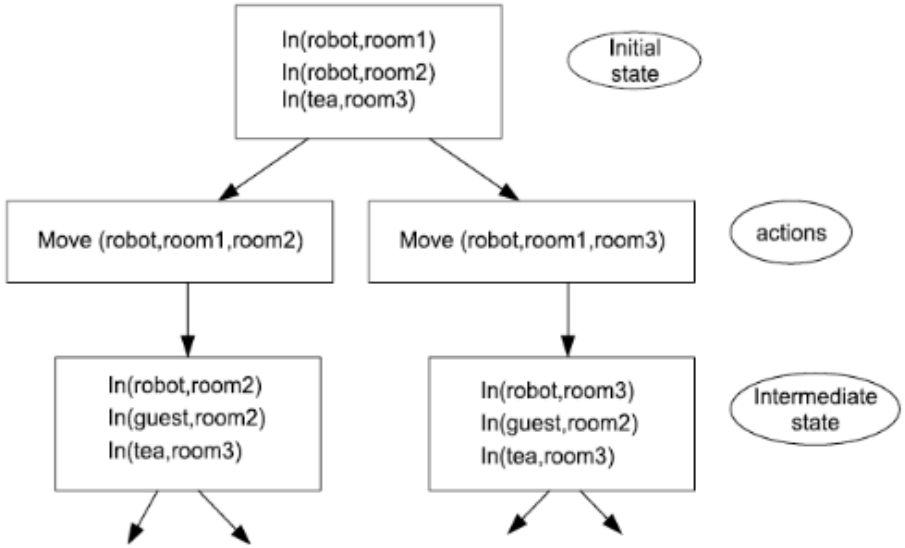

Figure 9.5 Progression planning.

The problem is formulated as below:

1. ***Start state:*** The start state is the initial state of the problem prior to any action applied to it. It assumes that the things not mentioned (objects not involved) are not true.

2. ***Goal check:*** Checking of whether the goal is achieved/reached or not reached.

3. ***Actions:*** It involves a list or sequence of actions with respect to the pre-conditions that allows moving to the next intermediate state.

4. ***Step cost:*** It is a cost associated with every action that is taken. Most often, this is 1.

Any of the search method that comes to the solution, if at all exists, is complete. Hence, a complete planning algorithm is possible to obtain. But are there any drawbacks of such a search? We can see in Figure 9.4, the scenario is simple with a limited number of propositions. But what will happen if the number of propositions is increased? It is also observed that the actions that are considered in the planning are all the possible actions. There is a possibility that irrelevant actions not leading to the goal are carried out. For understanding, let us take an example. You plan for getting 75% marks, and focus on some topics that are out of syllabus or some points are never asked. Since you are unaware of it, you study them too, and hence, they are also taken in the planning. If you have the knowledge, the plan can be made better. Here, heuristic comes in picture. Can forward planning be made more efficient with the aid of heuristic? We will discuss heuristic methods later in the chapter to clarify this. Let us begin with the backward state-space search.

Regression Planning

In case of regression planning, the reverse process is applied. To plan, the search is started from the goal state to the initial state. So, we move towards the intermediate (pre-states) states. The question is how can we reach the pre-states? Figure 9.6 depicts a snapshot of backward search. Let us see how the algorithm works.

1. Selection of an action that satisfies all or some of the propositions in the goal state.
2. Reforming a new goal.

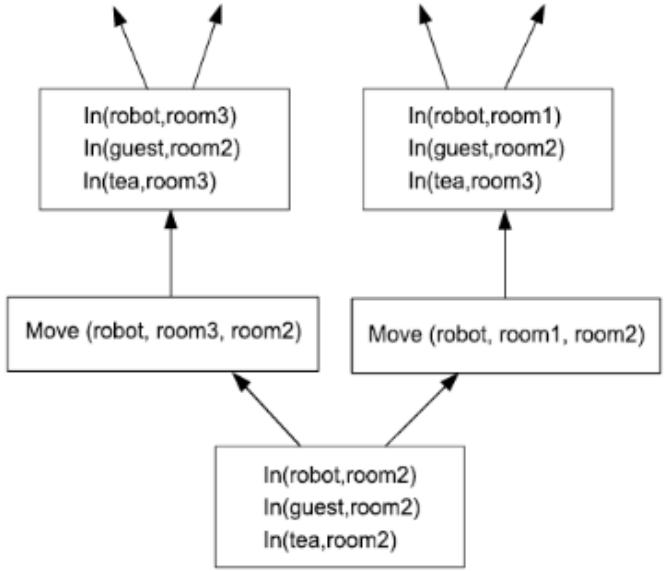

Figure 9.6 Regression planning.

Here, the goal propositions that are satisfied by invoking the actions are eliminated, whereas unsolved ones are keep intact. The pre-condition is the state prior, so this is added.

Steps 1 and 2 are repeated till the initial state or the start state is reached.

Note: The way the propositions are achieved is important here (i.e., ordering is important). The main advantage of backward search is that the relevant actions are considered.

Any of the search strategy can be made use of in the process of planning. Let us proceed towards the discussion of heuristics.

Role of Heuristics

The methods discussed above, i.e., progression and regression planning can be made efficient with the use of heuristics. Is it possible to identify the number of steps or the actions to be executed so that the goal is achieved? This comes to a NP-hard problem. Still, some solutions to it are discussed here.

So, what we intend to do is to find an admissible heuristic. This is done in the following way:

1. By converting the problem into a relaxed one (Here all the pre-conditions are removed.)
2. By assuming that the sub-goals are independent (So, the cost to solve the problem is approximated by summation of the costs, where the sub-goals are solved independently.)

To summarise, the progressive and regressive planning actions are dependent on the sequences. There is a need to have planners that can consider sub-goals, solve them independently and then the final plan can be a combinations of the sub-results obtained. Thus, we can proceed to partial order planning.

9.8 PARTIAL ORDER PLANNING

The concept of total order planner states that any planner that maintains the solution (may be partial too) as a totally ordered steps it has identified so far forms a linear planner. Whereas, if partially we are able to put up the ordering constraints (temporal), it is referred to as a *partial order planning*. With the progressive and regressive plannings that are linear in nature and are often referred to as *totally ordered*, there is a necessity to have the plannings to exploit the sub-problem structure.

Consider a case where you have to latch the door. With the use of lock and keys, the total order plan and partial order plan (POP) are depicted in Figure 9.7.

There is a notion of least commitment strategy in POP. *Least commitment strategy* is a policy, where the decisions or the choices are delayed. It simply says that 'do not make any decisions unless required'. One advantage of using this is that it avoids the re-work. The tasks might have to be undone. To understand the concept, take a simple example. Assume some assignment is given for a particular course. A student completes the assignment on the same day. But he does not get it evaluated from the course teacher till a deadline or say some grades regarding submission have been put up. (Hoping that the re-work would be avoided!)

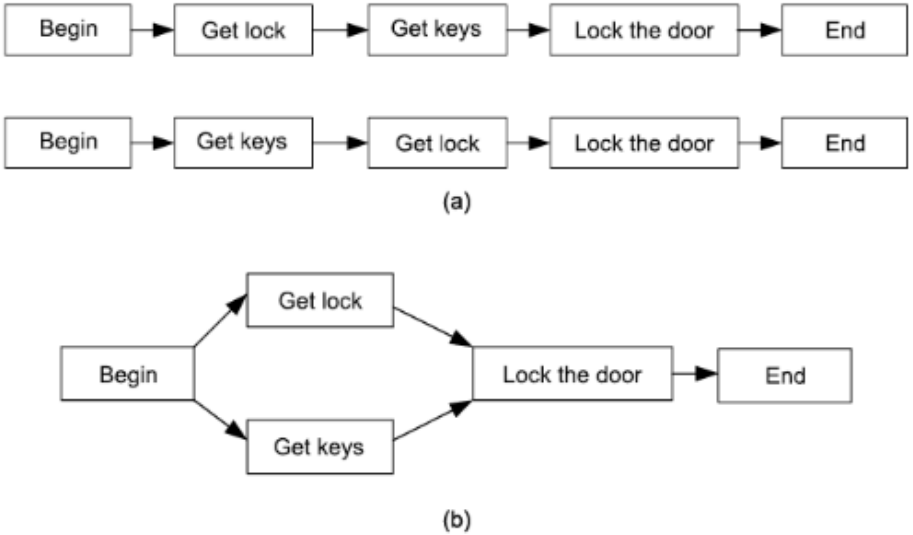

Figure 9.7 (a) Total order planning and (b) Partial order planning.

In case of partial order plans, the plans are created by making a search through plan spaces. It follows the least commitment strategy. This strategy allows to make choices relevant to current solvable part of problem. The question is how can the partial order plans be actually represented? The answer is in the form of graphs. Typical graph structure for the POP comprises temporal constraints. These constraints are the ones that specify that state x is before state y. In POP, the planning algorithm can put two or more actions into a plan, where the sequence in which the actions are to be carried out does not matter.

POP Representation

Consider a case of four states. A graphical representation for such a POP is shown in Figure 9.8. The temporal constraints are State 1 < State 2, State 1 < State 3, State 2 < State 3, State 3 < State 4.

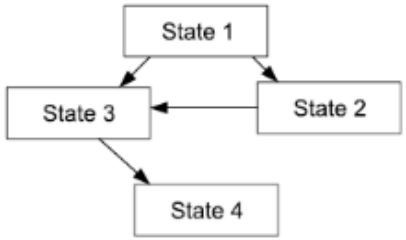

Figure 9.8 Basic graph representation for POP.

The total order plans here are (State 1, State 2, State 3, State 4) and (State 1, State 3, State 2, State 4).

We now look at the creation of the POP. We know that the plans are created by searching in the plan space. Let us create a graph. An arc shows the transition from one state to another. These arcs can be categorised into two types—where a normal transition from one state (plan) to another can occur, and where constraints indicating that a particular state has to occur before next are indicated (time factor is not accounted).

The plan is represented as follows:

1. Set of plan steps. The operators are mapped into steps.
2. It also has 'ordering constraints' for the steps. This is represented as $S \prec S'$, where S occurs before S' step.
3. Bindings on variable are also added of the form var = x, where var is variable and x could be referring to another variable on some constant.
4. The causal links are also established. It is represented as $S \rightarrow$ cl:S', where the pre-condition 'cl' for step S' is satisfied by S.

A *consistent plan* is defined as a plan that contains no cycles for ordering constraints and there are no conflicts with the causal links as well.

The partial order plan algorithm comprises

1. Begin with initial plan
   This plan consists of only start and finish steps.
2. Till a solution plan is achieved:
   (a) Select an unachieved pre-condition and achieve it.
   (b) Resolve threats by applying promotion or demotion.

Let us take an example to understand the threats, promotion and demotion. Consider a case, where you have to update on the social network sites or the professional ones like Linkedin and check office mails.

Writing it syntactically,

1. **Goal:** update(Linkedin)^update(facebook)^check(office-mails)^update(twitter)
2. **Initial step:** Login(office-email-id)
3. **Operators:** The operators can be
   (i) Checking-mails
   Pre-condition: Login(office-email-id)
   Post-condition: check(office-mails)
   (ii) Updating-social-network-sites
   Pre-condition:
   Post-condition: update(*x*)^~Login(office-email-id)
   (Assume you are already ready with your laptop and also the sites are on for you to update.)

Plan search: The first step is where we want to go from the initial state, i.e., to the goal state.

In the next step, as shown in Figure 9.9, a graph is generated. Here, the unsolved goals are just added. The links are established.

Moving ahead, there are some pre-conditions that we have not added like login to office email. So, add them. The next step is depicted in Figure 9.10.

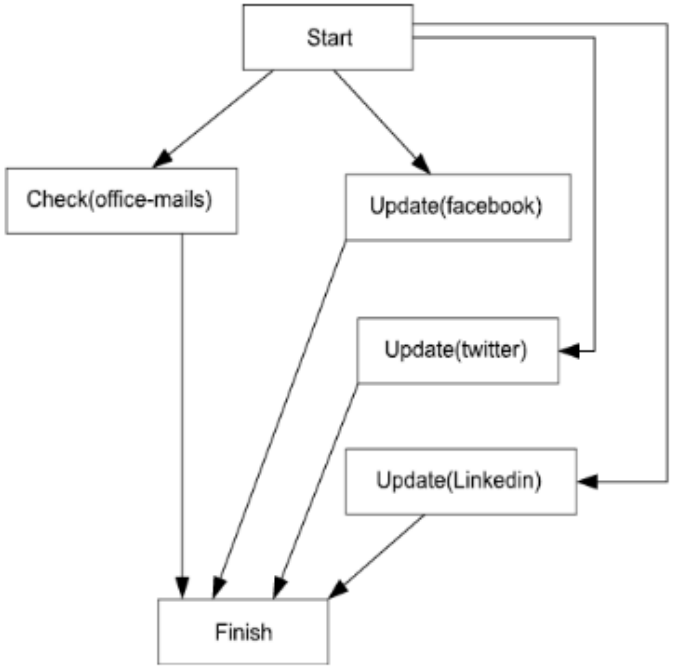

Figure 9.9 Adding unsolved goals—Step 1.

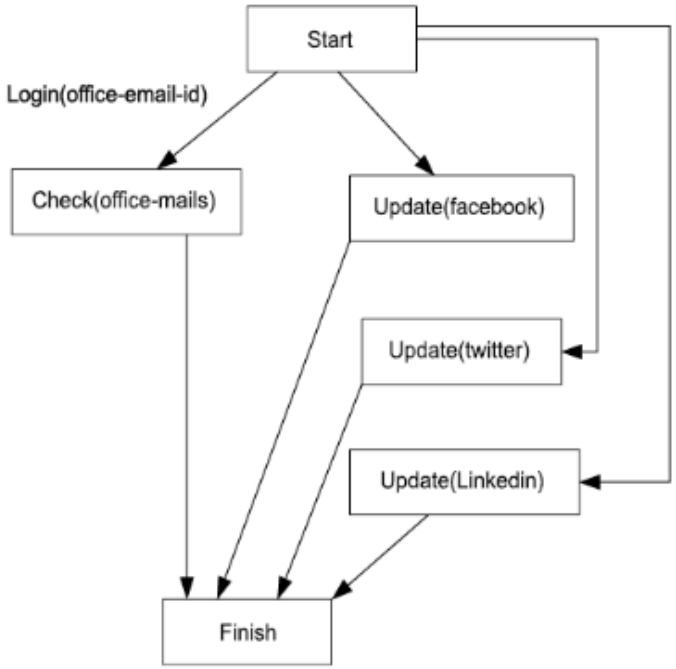

Figure 9.10 Adding pre-conditions—Step 2.

The next part is to add the ordering. As we can notice from Figure 9.9, we say that the four activities can be performed at any point of time. But the steps of updating the sites do not require the office-login. So, if they are performed prior, then this pre-condition will not occur. (In the given example, this is required.) Hence, it is treated as a threat. So, this is added where it is indicated that the update operation can be performed any time after the emails are done. So, the graph now becomes different as shown in Figure 9.11. Thus, an ordering constraint is added. It is said here that the previously achieved pre-condition has been clobbered. We will discuss this once the entire plan is complete.

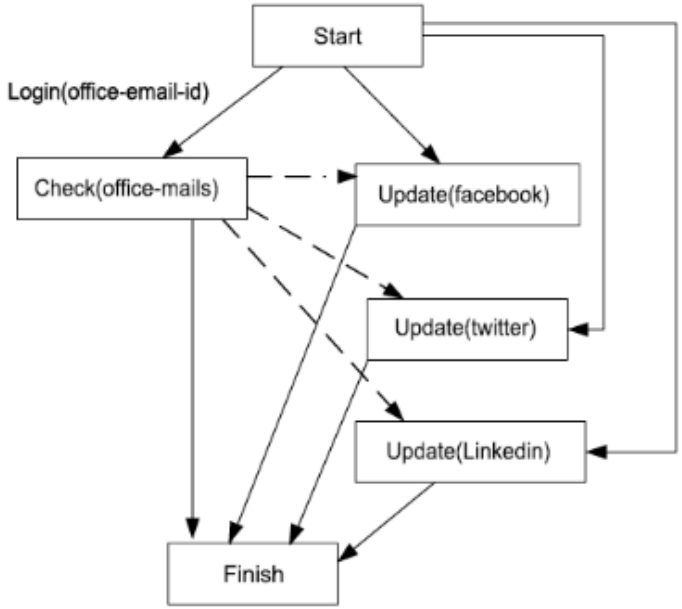

Figure 9.11 Adding pre-conditions—Step 3.

The ordering constraints are Check(office-mails)<Update(facebook), Check(office-mails)<Update(Linkedin), Check(office-mails)<Update(Twitter).

So, the solution we have obtained in step 3 is a complete plan.

Coming back to the discussion of clobbered, we have detected a threat in the previous example. The threat is fixed by promotion or demotion, i.e., declobbering.

Consider the following case of actions a_X and a_y

$$a_x \xrightarrow{e} a_y$$

where e is the effect of action a_X.

Now, there can be an action a_Z that could have an effect e'. In such a case, there is a threat. This is resolved by ordering a_Z after a_y or ordering a_Z before a_X. The former case is promotion and the latter case is demotion. So, in promotion, the threat step is ordered after the link, whereas in demotion, it is before the causal link. In the given scenario, the threatening steps were of update. So, the threat was allowed to come after the check-mails was done. Hence, we applied a promotion to it.

Let us represent the promotion and demotion as given in Figure 9.12.

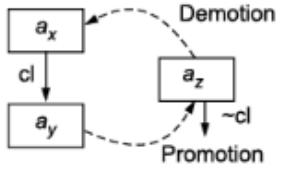

Figure 9.12 Promotion and demotion of action.

Figure 9.12 shows that a_y is generated from a_x . a_y has cl as pre-condition. In a POP, assume a step a_z is added, which has ~cl as the effect. If a_z occurs between a_x and a_y, then it creates issue. a_z is conflicting action. So, an ordering constraint of promotion or demotion, as shown in Figure 9.12, is added.

Similarly, there is a case of *open pre-condition*. If a state s has some pre-condition p that does not have any link (causal link to it is an open pre-condition), then we say that this is not solved yet, and hence, there is a need to solve it. For the same, establishment is done. By establishment, we mean that a new plan and/or a link is added. This can be carried out in the following two ways:

1. *Link addition:* It involves finding a step that is before s and has the pre-condition true. Then, a link is added from this existing step to 's'.

2. *Plan addition:* It involves adding a new plan. This new plan now contains the pre-conditions as post-conditions. Further, a link is set from this new plan to s.

While discussing POP, we need to look at its pros and cons too. It is clear that the POP is complete. Moreover, it is systematic also. With the use of least commitment strategy, it may have shorter searches. Addition of heuristics can make it more efficient. Now, can you think of some drawbacks of POP? What can you say about the search spaces? They are huge. This is due to the concurrent actions. Also, it is very difficult to state that what is actually true in a particular state.

9.9 PLANNING GRAPHS

After having an elaborated study on POP, we now come to planning graphs. The concept of planning graph was introduced by A. Blum and M. Furst in 1995. *Planning graph* is a data structure that encodes the information contained in the states regarding their reachability. The basic idea, as introduced by Blum and Furst, is to construct smaller size graphs that would contain partial information about the reachability. Reachability means an estimate about the steps required to reach a particular state from the initial state. Also the graphs can be used in heuristic estimation as well. By the use of algorithms like Graphplan, it is possible to get a direct solution from the planning graphs.

A *planning graph* is a layered graph comprising states and actions that form a layer and they appear alternately. So, it can be represented as follows:

{State 1, Action 1, State 2, Action 2, ..., State n, Action n}

For each operator o that belongs to action$_x$ (say), there is an edge from literal in state x. This literal is the pre-condition for operator o.

Constructing the Planning Graph

The graph is constructed in layers. As discussed earlier in STRIPS, it consists of states and actions forming one layer. To begin, the first layer of state consists of an initial state. So, every positive literal in the initial state and every negative literal not in the state are added. This forms state 1. For action 1, it consists of the operators for which the literals are the pre-conditions from state 1. For the next layer, the state 2 consists of all the post-conditions of the operators used in action 1. This process is carried out till there is a stabilisation. It means that the layer$_{n-1}$ is equal to layer$_n$.

Let us take an example of constructing a building and selling it. Let the initial state be that the building is constructed, i.e., constructed (building) and the goal state is constructed (building)^sold(building).

Let us have two actions—selling of the building and constructing it.

Action: Selling(building)
Pre-condition: constructed(building)
Post-condition: ~constructed(building)^sold(building)

Action: Constructing(building)
Pre-condition: ~constructed(building)
Post-condition: constructed(building)

The planning graph for the example is represented in Figure 9.13. Details are given below:

***Layer* 1:** It involves state 1 and action 1.

State 1: (All the atomic facts are in initial state) union (negation of the atomic facts is not in the state). So, state 1 has constructed(building) and ~(sold(building)).

Action 1: All the action where the pre-conditions are satisfied, so it will have only selling(building).

***Layer* 2:** This involves state 2 and action 2.

State 2: We map the post-conditions here. The literals that remain unaffected are carry forwarded.

Action 2: Similar to action 1, it carries the task.

***Layer* 3:** It involves state 3.

State 3: As it is observed that the state is similar to state 2, this is called stabilisation. This stage comes after n iterations (say), where the states remain same. Hence, there is no

need to go ahead for building the layers.

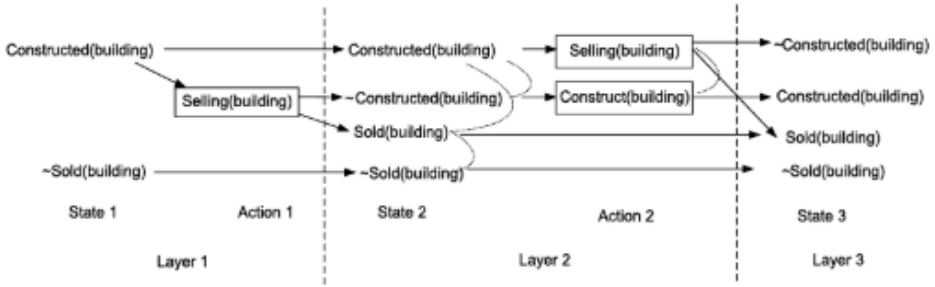

Figure 9.13 Planning graph.

One more point to address here is that we can see some dotted arcs in the states. These are called as *mutex links*. They indicate that they are mutually exclusive. This is due to the conflicts that arise. This means they cannot be selected together, or else there would be a conflict. This pair in conflict is called *mutex*. (*Note:* Only few mutexes are shown in the diagram). The mutex can exist for literals as well as for actions.

With respect to the actions, it is said that the two actions at the same layer are mutex if they show the following conditions:

1. *Inconsistent effects:* In the sense, say actions a_X and a_Y are mutex, if a_X results in L_X and a_Y results in $\sim L_X$ as post-conditions.

2. *Interference:* A pre-condition of a_X and post-condition of a_Y are inconsistent. That is, Pre-condition = ~Post-condition.

3. *Competing needs:* In this case, the pre-conditions are actually the inverse. Say pre-condition p_1 in a_X and p_2 in a_Y are competing, i.e., $p_1 = \sim p_2$.

In case of literals, the literals (or the states here) are mutex if

1. There exists inconsistency, i.e., if l_1 and l_2 are the literals then $l_1 = \sim l_2$ and $l_2 = \sim l_1$.
2. If there exists a pair of actions a_X and a_Y such that l_1 is the post-condition of a_X and l_2 the post-condition of a_Y, and a_X and a_Y are mutex, then l_1 and l_2 are also mutex.

Heuristics

Planning graphs are very useful in terms for heuristic estimation. So, we want to estimate the cost. A very simple thing to understand is that suppose a literal does not exist in the final layer and it is present in the goal (This is after the planning graph has stabilised), then the goal cannot be achieved. Further, in heuristics, the numbers of levels are counted to be the cost. But will this be a better option? No. The reason being the number of actions that take place in each of the layers. But if a condition is added where only one action can occur at a particular time, then this is called *serial graph*. A serial graph also adds the mutex. The cost to get the literal in the goal is computed as the summation of cost of the number of levels it appears before.

Why is it necessary to have all this explanation regarding literals and layers? Where is it useful? With the planning graph, we are able to identify the reachable states. The reason for constructing the planning graph is to get the heuristics and a plan extracted from the planning graph. So, now we discuss the plan extraction.

How can we extract a plan? Through the use of Graphplan algorithm, we can carry out the process of plan extraction.

One point to remember here is that the plan extracted from the planning graph can be longer than the actual layers in the graph. The algorithm first generates the graph and then extracts a plan from it.

The algorithm is briefed below:

1. Since the aim is to extract the plan, it first checks if the literals (here, the states) in the goal state are present in the current layer with no mutex chains.
2. If so, there is a strong probability that the solution exists. The algorithm tries to get the plan extracted.
3. But if not, then it expands to the next level. So, it goes to the next layer. This is done with the addition of actions and states.
4. The steps 1, 2, 3 are repeated till a solution is found or it is observed that the solution will not exist.

While extracting the plan, the process is carried out from the last layer to the beginning of the graph. The actions are selected that do not have the mutex links and also there should not be any mutex links in the pre-condition as well. So, we can say that the process is backward. This process is carried out till all the required goals are reached or rather they are satisfied.

When will the algorithm terminate actually? It will be based on the mutex. For example, if in the goal, the states are mutex in the planning graph, then the solution is not going to occur. As discussed before, there exists a stabilisation constraint, where the layers become same or identical. But still, there can be further expansion of stages if the solution is not obtained; hence, the termination of the algorithm is a complicated issue.

9.10 HIERARCHICAL PLANNING

The traditional planning approaches always have to face complexity issue. They lack the

ability to build a structure that can distinguish and prioritise between the important and unimportant aspects., i.e., operators and properties. Hierarchical planning overcomes this inability by providing abstraction in planning.

The abstraction is provided at two levels.

1. Situations and 2. Operators

1. Situations: Weight is assigned for each literal, where priority is given to literals with higher weights. For example, weights can be assigned as

Property: Weight
On: 3
Clear: 2 and so on.

2. Operators: In order to provide operator abstraction, any operation is viewed as operator abstraction of other operators, say for example, for 'lift' operator, we can split it as

(i) Placing the robot arm
(ii) Holding the block
(iii) Picking the block

This planning is demonstrated by ABSTRIPS. The hierarchical planning is governed by criticality value. Any operation that possesses minimum criticality is trivally achievable. This specifies operators with few or no pre-conditions, whereas one with more pre-conditions has high criticality. This type of planning is effective where pre-conditions are shown to be achievable and can be achieved independently.

9.10.1 Hierarchical Task Network (HTN)

Some recent work on hierarchical planning and co-ordinated plan executions by P. Gorniak and I. Davis discussed the need for hierarchical planning, where it put forth that the STRIPS are turning out to be poor designers in planning. With an approach based on hierarchical task networks, the paper addressed the co-ordinations and efficiency along with planning intelligently. A complier based on JSHOP, a hierarchical planner is used. The idea presented is to write a tool that generates the specific plans. It shows its efficiency in case of generation of plans for combat situations, squad as well as in game context. Some more examples of HTN are SIPE-2 or SHOP.

What does this HTN planning actually do? The HTN planning puts to practice this abstraction making use of the operators. The operators can be abstract and primitive. But the final plan generated is only with the primitive operators.

A simple HTN is depicted in Figure 9.14.

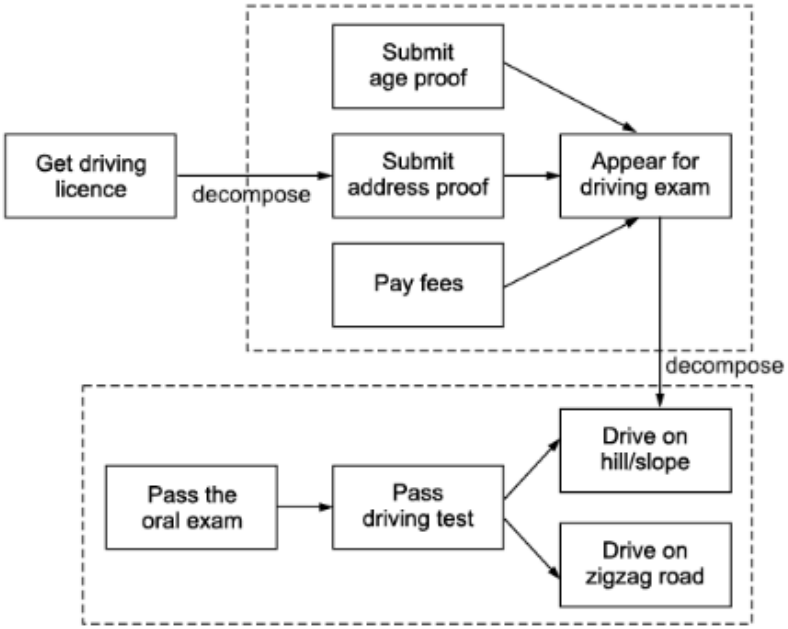

Figure 9.14 A simple HTN.

To summarise, in hierarchical planning, the actions are split into sub-tasks. The sub-tasks are specific. The actions are serialised into an abstract level. So, we are, in short, prioritising the tasks, where the operators are actually planned. Most often, HTNs are used, where the problem structure itself has some organised hierarchy. In case of HTN, rather than having the main goal set up, it comprises partially ordered tasks to take up. So, a decomposition process starts till the planner can reach the primitive tasks that can be easily carried out with the operators. Whereas, for the non-primitive tasks, further decomposition takes place.

9.11 NON-LINEAR PLANNING

A plan that consists of sub-problems, which are solved simultaneously is said to be a *non-linear plan*. In case of the goal stack planning, as discussed previously, it poses some problems. To achieve any goal, it could have an impact on the one that has been already achieved. In linear planning, the effects change according to the situations in which they are operated. So, just one goal is taken at a time and solved completely before the next one is taken is the job of linear planning.

Let us take an example. You want to take the car for servicing and have to make an

important phone call. To complete these two tasks, we can have an interleaving form. Rather than completing both the tasks in a linear way, after completion of the task 1, as partial step, i.e., start the car and put on the Bluetooth, then complete the task 2 of phone call and then finally, complete the task 1 by leaving the car at the service station. This can be an example of non-linear planning. Figure 9.15 depicts a simple non-linear plan.

Figure 9.15 Non-linear planning.

There is a concept of constraint posting that comes with non-linear planning. The constraint posting states that the plan can be built by

1. Addition of operators or suggesting operators
2. Ordering them
3. Binding the variables to the operators

Initially, we are not aware of how the operators that are available should be worked on. So, incrementally, the constraints and the details are added. Planning systems like TWEAK and MOLGEN are non-linear in nature, which employ this constraint posting.

9.12 CONDITIONAL PLANNING

Before going into discussion of conditional planning, we look at undeterministic environment and prepare some background for the concept. The methodologies that have been studied belong to the classical planning. Here, the environment is deterministic, fully observable and static. What can happen in case of partially observed environment? For example, you have planned for studying in the preparation leave. For a particular subject, you are to go to the market to purchase the book. You decide that on a particular day, say Tuesday you would go and get the book. But on that day, all of a sudden, the shopkeepers call a strike indefinitely. Your entire planning is disturbed and so is the plan. Hence, we can say that the information available was incomplete (owing to sudden change) in the plan. In such a case, the plan generated is actually correct, but due to the circumstances, it turns out to be less strong. Take one more case, where the plan is hampered if the exam schedule is changed. In all these scenarios, there is a need to handle the incompleteness.

Now, we look at the concept of bounded indeterminacy. This governs how much concrete the environment is in which an agent has to operate. A *bounded indeterminacy* is defined as a condition in which actions can result into unpredicted outcomes, but it is possible to get the listing of the possible outcomes. For instance, if you visit a person without intimation, then there is uncertainty that the person may be at home or may not. Hence, there is unpredicted outcome from the known set. So, planning is very much possible with this bounded indeterminacy. In conditional planning, it handles this indeterminacy. A simple approach to this is to construct the plan with the possible outcomes. So, the possible alternatives are listed. So, in case of above example, you can have plan for both outcomes. In case the person is at home, you can have a meeting with him, else you can watch a movie in a theatre next to his residence. We can say that with the rules of if-then, conditional planning occurs. Further, it also makes use of AND-OR graph search for the planning. Conditional planning can occur in fully observable as well as partially observable environment. In case of fully observable environment, the agent in operation is aware of the current state. Hence, the planning becomes simple with the rule-based structure. In partially observable environment, since the entire information is not available, planning becomes difficult. In such cases, the conditional planning makes use of belief states. Considering the same example, after reaching the person's residence, you came to know that he was not there. So, as per plan, you go for the movie, but there is a possibility that movie tickets may be available or may not be. This state is the belief state. It is defined as a set of possible states an agent can be in. With the use of the belief states and AND-OR graphs, the planning in partially observable environment can occur. The major drawback of conditional planning is that they can be harder than the NP-complete problems. Still, what can you say about the space requirements of this approach? They are large enough.

Conditional planning is also referred to as *contingency planning*. So, it is a planning approach, which can handle the various contingencies that can arise. These types of planners belong to the category of non-deterministic domains, where a dependency lies on the parameters other than the current state.

9.13 REACTIVE PLANNING

Reactive planning is planning under uncertainty. Here, rather than building the plan with the branches, it makes use of the if-then rules. So, what different are they doing? The reactive planners are based on the concept that they should be able to handle an unknown situation too. So, the reaction rules are used that help them in doing so. But there is also a need to have reasoning while applying the reaction rules. The reactions rules are prioritised. A rule selection is based on the priority and a holding condition that maximises the priority. The rule which is at present in execution is said to be active whereas the ones with holding priority (we can call them possible competitors) are pre-active and the others are *inactive*. Many's the time, in reactive planning suspension can occur in the rule that is at present in execution. A B-tree structure is used in reactive

planning, where the algorithm selects the rule. Sometimes, no rule can be selected. In such a case, things are dependent on the algorithm implementation for rule selection.

9.14 KNOWLEDGE-BASED PLANNING

The planning methods (like HTN) actually belong to the category of knowledge-based planning. Early in 1998, Kautz and Selman identified the planning knowledge categories as follows:

1. Knowledge regarding the domain.
2. Knowledge regarding the good plans
3. Explicit search control knowledge.

It is not just this knowledge, but today, it is also about interacting with the user and getting the preferences. Some of the features of a knowledge-based planner are mentioned below:

1. *Expressive:* It should be able to express the types of goals and constraints, to look at the additional resources, and so on.

2. *Correct:* The HTN planners have proved to be effective in terms of correctness. The property of correctness tells that though the real-world domain cannot be modelled, system can be verified with the effectiveness of some properties.

3. *User interaction:* A planning model can be made more effective, if it allows for user interactions. This allows the necessary changes to be made in the plan as per the requirements and provides guidance in the search too.

4. *Constraints:* With ontology, it is possible to encode the knowledge, and thus, reason about the system. This proves the method to be more effective, as it is able to reason that the relations set earlier cannot be changed. Further, temporal constraints that are time-based help in developing a better planner.

9.15 USING TEMPORAL LOGIC

The planning is based on the time in case of temporal logic. The rules and the propositions are governed by the time factor. It tackles conditions that would gradually be true, or dependency of one condition to be true based on some other, and so on.

Temporal logic essentially provides a mathematical framework that can be made use of in the reasoning process during the planning. The requirements of the system can be represented in a better way by the use of temporal logic. This results in the correctness of the process. After studying the knowledge-based planning, do you think that there could be a way to use temporal logic here also? Yes, it is very much possible. Moreover, domain dependent search control knowledge can also be represented with the use of temporal logic. This is further exploited and used in forward chaining planner. TLPlan is an example of such a system, where the temporal logic is used. Currently, it is put to practice in the motion planning.

9.16 EXECUTION MONITORING AND RE-PLANNING

As the name suggests, *execution monitoring* is the process of checking the current state and its percepts, and ensuring that the things are moving ahead as per the plan. When things are not going as per the plan, re-planning is done.

Let us take the example of a robot serving tea to the guest (refer to Figure 9.2). We have a plan, according to which the robot has to move to room 2 to give the tea to the guest. Now, while executing the plan what will happen if the guest moves to room 1 in the mean time? Here, comes the execution monitoring that performs the check on the current percept. So, the pre-conditions are checked. This check indicates that the guest is not in room 2. This is handled at the time of execution. Hence, re-planning for the set of actions is done next to some point in the plan. With the agent scenario in picture, this type of planning involves two types of agents to control the execution process—Action-check agent and re-planning agent.

It is the action-check agent that checks if it is possible to execute the next action. If so, it, in turn can direct some work to the plan-check agent that would verify the plan remaining. The job of the re-planning agent is to maintain a track of the plan completed till the current state and the one remaining. It is the re-planning agent that actually calls the action-check agent, and further, based on discrepancies, re-planning of the actions takes place.

The algorithm for this is summarised below:

1. Observe the current percept.
2. Check for the pre-conditions.
   If satisfy, proceed.
   Else report a failure state and re-plan.
   (i) Find a state in the plan such that the failure state is repaired.
   (ii) Create a new plan with this new state added and continue.

9.17 CONTINUOUS PLANNING

Continuous planning can be thought of as a planning, wherein the monitoring and the re-planning activities are clubbed together. In case of these planners, the operation is incremental in nature. By incremental, we mean that the plan formulation is dependent on the current percept, and the continuous planning agent always begins with this percept and generates a plan. So, at each point of time, the percept

is observed and the plan is generated. Thus, the methodology is incremental.

Considering the same scenario of the agent and the guest, if the plan for the next action is selected based on the current percept available, it would prove to be a better option. So, if the agent has planned to execute the action of 'give tea', with pre-conditions as guest and robot both are in room 2, the continuous planning agent sees the change, whether the guest has moved to room 1. If so, the plan is changed. The agent in this scenario detects that the pre-condition has turned out to be false. Hence, the other actions are required to be worked out. These types of planners are similar to that of POP, yet they need to carry out a lot of activities like detecting conflicts in case of action execution, detecting unwanted actions, updating the new actions, building and identifying the plans, and so on.

9.18 MULTI-AGENT PLANNING

The multi-agent planning involves use of multiple agents to carry out the planning tasks. The background behind this is that a teamwork can prove to be better when an individual cannot accomplish the task. We are already familiar with what planning does and what a single agent can do. If the task is extended to multiple agents, it would be easier. The multi-agent planning generally is of two types, i.e., by and for multiple agents. The planning in case of agent environment obviously needs the interactions between them; hence, there should be co-ordination and communication. This is necessary from the goal achievement perspective. What would happen if the agent would work with only his available set of environment? Can the goal achieved be efficient? Can an agent depend on the other agents actions? The answer to all these questions is yes. Hence, there is a need to have proper co-ordination between them. At the next level, it could even be distributed.

Can there be any issues that are required to handle in this planning? The answer is definitely yes. There are things that need attention like robustness of the plans, the costs involved in the interactions along with the scalability issues.

Where is this required? The multi-agent planning is most commonly used in the industries associated with assembling of components, in case of electronic devices or vehicle assembling and not to forget, the gaming part as well.

The tasks involved in multi-agent planning technique are summarised below:

1. Setting goals and assigning to individual agents
2. Refining the goals and further decomposing them into sub-goals
3. Converting sub-goals into achievable tasks
4. Scheduling the tasks (This includes resource allocation to agents and timing deadlines, if applicable, for the task)
5. Co-ordinating and communicating to avoid conflicts
6. Executing the plan

9.19 JOB SHOP SCHEDULING PROBLEM

It is a problem that needs to handle the time constraints, the scheduling, and at the same time, has to perform check on the availability of resources. Let us understand how it is resolved with planning.

The job shop problem consists of jobs j_1 to j_n that are to be completed. To complete them, machines m_1 to m_n are used. For the jobs to complete, there is a sequence of actions, say a_1 to a_n that is required to take place on the machines. Every action has a specific duration and it might make use of some resources also. The problem is to minimise the time required for the completion of the jobs, and at the same time, handle the resource constraint, if any. (Remember that there are some variants for the job shop problem.)

So, is it a scheduling problem in planning? Yes. The job shop scheduling problem occurs in cases of the production scheduling. There is a need to have an appropriate planning so that the constraints are taken care of. Let us take an example, where the scheduling is to occur for two vehicles to assemble the engine assembly, frame assembly and the chassis. A POP is shown in Figure 9.16. The numbers above each of the action indicate the time it needs to execute. While scheduling, there is a concept of slack. Suppose along with the time durations required to get the action completed, if early start and late start timings are provided, then this slack is identified. Let us say that the early and the late start timings for action 1 of engine assembling for both the vehicles is 0,0. Then, there is no slack. But suppose the early start time and the late start time for action 2 for vehicle A is 40,50, then there exists a slack of 10 for the action.

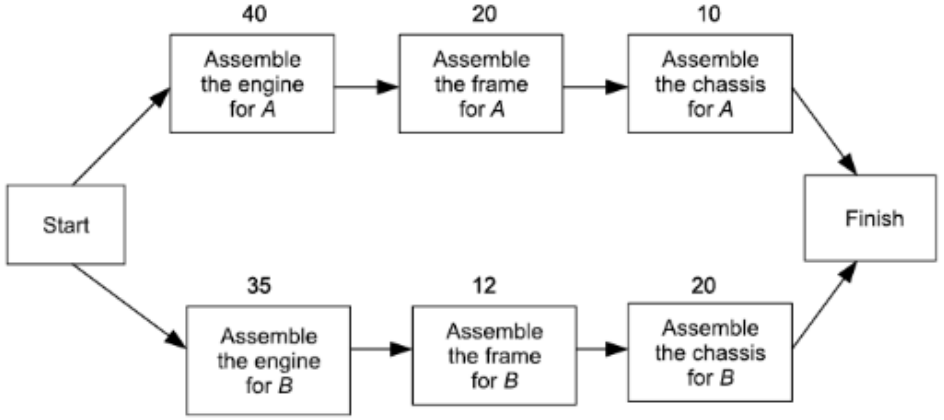

Figure 9.16 A partial order plan for the problem.

This can be very much represented in terms of Gantt chart. But we are here to talk about planning and constraints, let us take things ahead in the constraint part. Suppose for each of the action, there is a resource person, who actually evaluates and monitors whether the things have been assembled in a proper way or not, and then, there is a constraint added to it. If there are six different people to carry out the task, the constraints are nil. But if only three persons are available to carry out the fixed tasks like one working as engine checking manager, one as frame check managers and one as the chassis checking manager, things would get very much complicated. There is a need to have a proper plan for this, wherein these resource persons should be used in the planning process with the constraints that exist. The solution to this is depicted in Figure 9.17. The time required here comes to 107.

Can there be any heuristics in case of planning for the job scheduling? The greedy approach is the one that is often used. According to a simple rule that is applied here, the entire problem should be split into sub-tasks and then the planning and execution should occur. The actions can be very well-formulated using STRIPS. Yet, it is worth to mention that the constraints naturally add up the complexity of the problem, making it an NP-hard problem.

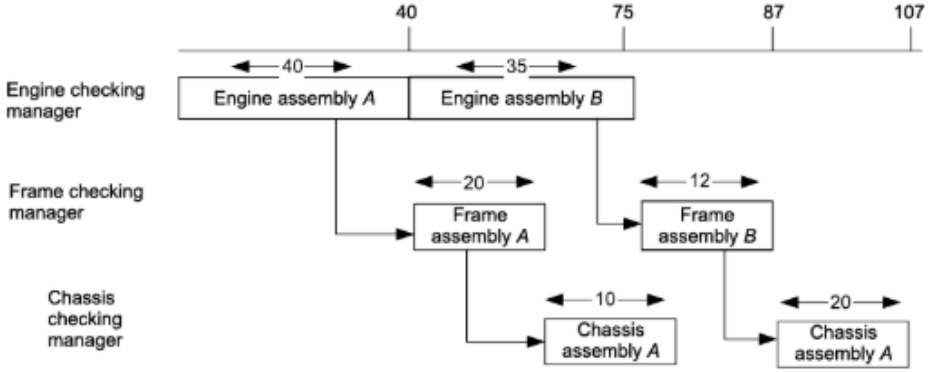

Figure 9.17 Possible solution for the problem with the resource constraint.

SUMMARY

With the necessity for the planning in AI, the chapter highlights the need and importance of planning in the fully observable environment as well as under uncertainty. A planning system comprises initial states, and goal states and actions to achieve them. The actions are represented in the form of operators to achieve the goal. Representation for a planning system has to be effective. Different languages like STRIPS, ADL are used for planning. Approaches like the goal stack planning and the means-ends analysis show their characteristics with the STRIPS. The state-space search and the forward and backward searches assist in the selection of an action. Addition of heuristics makes the planning more effective. Planning is not just concerned with the organisation of tasks, but forms the basis for reasoning. POP approaches prove to be a better option than the total ordering, helping in building partially and parallel executable plans. The least commitment strategy adds benefits to the POP.

Further, the planning graphs, helps in determining the reachability of a state and the term of mutex tackles the constraints. Hierarchical planning, conditional planning and the concepts of non-linear planning too are important when it comes to handle the real-time complex problems. HTN planning is been widely used in research at present for gaming purposes, whereas the multi-agent planning finds its place in military tasks and vehicle assembling.

KEYWORDS

1. **Planning:** It determines how and when to do/perform the necessary steps for defined goals.
2. **STRIPS:** It is a planning language that makes use of first order predicate, allows function-free literals and has historic importance.
3. **ADL:** A planning language that is more expressive than STRIPS and used for reasoning purposes.
4. **PDDL:** A planning language that is a superset of STRIPS and ADL and used for classical planning tasks.
5. **Goal stack planning:** It is an approach used in STRIPS that maintains stacks of goals to achieve the tasks.
6. **Means-ends analysis:** It is a rule based concept used in STRIPS that makes use of difference table for the selection of an action.
7. **Progression planning:** This planning starts with the initial state and proceeds with the search. The forward state-space search approach is used here.
8. **Regression planning:** This planning works from the goal state towards the initial state. The backward state-space search is used here.

CHAPTER 12

Natural Language Processing

Learning Objectives

- To understand the concept of natural language processing
- To study the techniques involved in the processing of the text or information
- To understand the analysis of the text and its applications
- To understand the importance and necessity of effective information retrieval systems
- To identify and study the various aspects of natural language processing, with the role of grammars in the representation
- To study the applications of natural language processing in building intelligent systems

INTRODUCTION

"We are like a pre-paid card with limited validity. If we are lucky, we may last another 50 years. Do we really need to get so worked up? It is ok to bunk a few classes, goof up some interviews; we are people, not programmed devices[1]. A part of speech by Chetan Bhagat on 'keeping the spark alive in one's life.' He means to convey a message by this speech, i.e., enjoy the life and not to take it too seriously. This inference is possible because you have understood the language as well as the context and are able to read between the lines. So, you have processed the language to get something out of it.

Even while reading the text of this book or while listening to a professor, one tries to understand, correlate, analyse or possibly map the knowledge that is written or verbal. So, the brain actually performs the task of these mentioned things. But when it comes to machine, we need to provide the intelligence—the intelligence to map and understand, which would enable it to analyse the extracted knowledge the way we want. Let us make things simple more technically!

What is natural language? A language that is in use and has been developed naturally. It is in practice and a way for communication. So, it can be in oral or written form. What are we going to study in natural language processing (typically referred to as *NLP*)? Let us try to bring it in more clarity. We can say that there are two things—one is the source in which the language lies and the other is the destination or what the processing needs to do as a target. So, NLP involves the study of various methods and approaches. This chapter discusses about the processing tasks, which can be carried out on the input language, and represent the language in some resulting form, as required. Hope now things are getting simple. So, to have some definition, *NLP* can be defined as

A computerised mechanism involving computation techniques to analyse and represent the text (oral or written) in some form for achieving human-like processing for a variety of applications.

Representation into some other form is a processing that includes the task of converting or mapping into some other suitable form. Even when we say that the language form is oral or written, one must be clear that things get very much complicated with the use of oral form owing to the noise in the signal.

NLP: Brief History

The work related to NLP was started with machine translation (MT) in 1950s. It was Allen Turing who proposed what today is called the *Turing test* in 1950s. It is the testing ability of the machine program to have written conversation with human. This program should be written so well so that one would find it difficult to determine whether the conversation is with a machine or it is with the other person actually. During the same period of 1950s, cryptography and language translation took place. Later on, syntactic structures came up along with linguistics. Further, the sentences were considered with knowledge augmentation and semantics. In 1960s, ELIZA (the most common NLP system) was developed that gained popularity. It was the simulation of a psychotherapist. At a very later stage, it was the case grammars that came up. Now, there has been a complete revolution in the NLP, with the machine learning approaches coming up. Many NLP systems have been developed till today and a lot of competitions are being organised that are based on the Turing test. Figure 12.1 gives a few of the tasks that come under NLP.

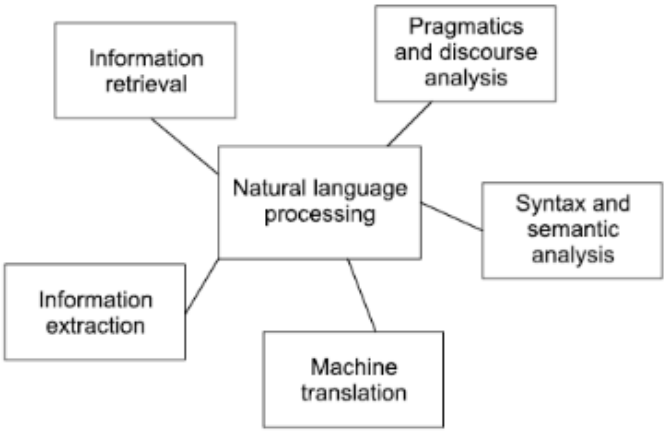

Figure 12.1 NLP: Tasks.

More than having the concept of natural language processing, natural language understanding is what that is essentially required. There is a very fundamental question—is Google or any other search engine making use of NLP, after looking at the concept discussed so far? What is your answer? Indeed, it does. Let us study about it too in the chapter.

12.1 LEVELS OF NLP

Before we go into the details of how actually the processing takes place, let us look at the different levels or the models (given below) at which NLP can take place and then study them in detail.

1. ***Morphology:*** It is the analysis of individual words that consist of morphemes—the smallest grammatical unit. Generally, words with 'ing', 'ed' change the meaning of the word. This analysis becomes necessary in the determination of tense as well.

2. ***Syntax:*** Syntax is concerned with the rules. It includes legal formulation of the sentences to check the structures. (Some aspects are covered in compiler's phase of syntax analysis that you must have studied). For example, 'Hari is good not to.' The sentence structure is totally invalid here.

3. ***Semantic:*** During this phase, meaning check is carried out. The way in which the meaning is conveyed is analysed. The previous example is syntactically as well as semantically wrong. Now, consider one more example, i.e., 'The table is on the ceiling.' This is syntactically correct, but semantically wrong.

4. ***Discourse integration:*** In communication or even in text formats, often the meaning of the current sentence is dependent on the one that is prior to it. Discourse analysis deals with the identification of discourse structure.

5. ***Pragmatics:*** In this phase, analysis of the response from the user with reference to what actually the language meant to convey is handled. So, it deals with the mapping for what the user has interpreted from the conveyed part and what was actually expected. For a question like 'Do you know how long it will take to complete the job?', the expected answer is the number of hours rather than a yes or no.

6. ***Prosody:*** It is an analysis phase that handles rhythm. This is the most difficult analysis that plays an important role in the poetry or *shlokas* (chants involving the name of God) that follow a rhythm.

7. ***Phonology:*** This involves analysis of the different kinds of sounds that are combined. It is concerned with speech recognition.

Can the analysis levels discussed be overlapped or interrelated? Yes. It is very much possible to have an analysis actually forming a fuzzy structure. They can work in stages, where the second level makes use of the analysis or the outcomes of the first level. We now study them in detail.

12.2 SYNTACTIC AND SEMANTIC ANALYSIS

Morphology deals with word formation from small parts. Morphological analysis finds out the pieces which have contributed for word formation.

The morphological analysis is simply separating or tokenising the words. Removal of the additional words like 's', 'ed' (as discussed earlier) and assigning them to the syntactic category are carried out in morphological analysis. So, we do not go into a detail discussion on it. (This sounds similar to the tokenisation concept in compilers!). We proceed to the next step of syntactic and semantic analysis. Let us begin with syntax analysis.

12.2.1 Syntax Analysis

Syntactic analysis is concerned with checking of the syntactic structure and its components. Once the words of a sentence are categorised into different parts of speech, the syntax analysis generates a parse tree. A parse tree is a representation for the sentence into a structure. This structure, which is hierarchical in nature, helps in identifying the sentence bits on which the next level of semantic analysis operates.

What things are essential to produce a parse tree? There exists the concept of grammar. A *grammar,* as is used in English, tells us about how the statements should be constituted, with the usage of the parts of speech at appropriate places. In NLP, grammar is the

description of language. It comprises rules. There is a Chomsky hierarchy that represents the different grammars that are used. Amongst them is context-free, context-sensitive and so on. Let us discuss context-free grammar.

Context-free Grammar (CFG)

A CFG consists of terminals, non-terminals, start symbol and production rules, where left side of a production rule contains only one non-terminal. The grammar describes the language. Following are the things contained by CFG:

1. *Terminals:* Terminals are syntactic symbols that cannot be replaced further. The terminals are written in small letters.

2. *Non-terminals:* Symbols that are replaced by terminals are called non-terminals. They are written in capital letters.

3. *Start symbol:* It refers to starting of the sentence rules (mostly it starts with capital letter S).

4. *Production rules:* These are the rules to be applied for splitting the sentence into appropriate syntactic form for analysis. They consist of terminal, non-terminals and the start symbol.

The grammar (in technical terms) is a quadruple, represented as (V, T, P, S), where V is the set of non-terminals, T is the set of terminals, P is the set of production rules and S is the start symbol.

For example, a sample CFG is

$$S \rightarrow \alpha_1|\alpha_2|R$$

$$R \rightarrow \alpha_3$$

Here, S and R are the non-terminals, whereas α_1, α_2 and α_3 are the terminals. What is the purpose behind using the CFG? It is used to check if a sentence is syntactically correct. Starting from the start symbol, if we are able to derive the sentence using these rules by substitution, syntactically correct sentence can be determined. So, the grammar is nothing but the rules that are required to from the hierarchical structure. How is this grammar used? Let us take an example that is always given for an English statement.

Let S be the sentence. This sentence can have subject and predicate. The subject can be a noun phrase as generally called and a predicate comprises the verbs/adverbs and so on. The grammar representation is discussed below for an example, say 'Ram read the interesting book' or 'Ram ate the delicious cake':

Let S : Sentence $\rightarrow$ *SB VP OB*
SB : Subject $\rightarrow$ *PN*
VP : Verb phrase $\rightarrow$ *ADV V*|*V*
OB : Object $\rightarrow$ the S_1
S_1 : Subset of S $\rightarrow$ *ADJ*|*N*
PN : Proper noun $\rightarrow$ Ram
ADJ : Adjective $\rightarrow$ interesting|delicious
N : Noun $\rightarrow$ book|cake
V : Verb $\rightarrow$ read|ate

Parsing: Once we have the grammar, parse tree is generated. One point to be noted here is there can be different sets of grammar rules depending on the text. For the above example, let us see how the parse tree is generated.

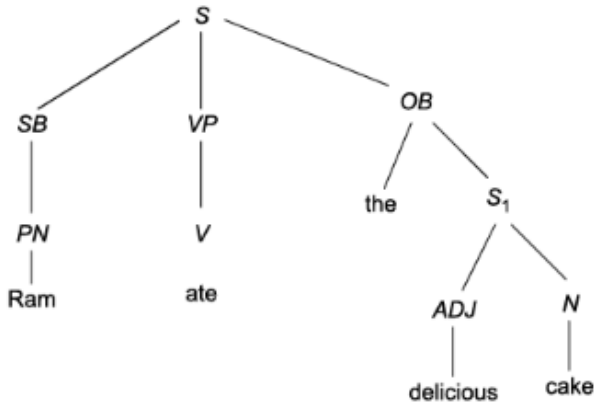

Figure 12.2 A parse tree.

In the process of generation of a parse tree, the grammar rules are compared with the input. Whenever a rule matches, it is expanded, thus generating the tree. As it is observed from Figure 12.2, at every stage, one rule is applied, thus helping in generating the entire structure.

Top-down and bottom-up parsing: Two ways of building a parse tree are top-down and bottom-up. In top-down, the parser begins with the main symbol or the start symbol(S) in the rules. It keeps on applying the rules till the terminals become the symbols of the sentences. Whereas, in case of bottom-up parsing, the reverse procedure is applied. Beginning with the sentence, the rules are applied backwards by replacing the words in the sentence with the appropriate rules till start symbol is reached.

Though top-down parser can build trees without the time factor making a constraint, it is quite possible that the trees are produced without having a look at what the next symbol or token in the sentence is to be parsed. So, may be wrong rules are selected. The branching factor is also a point of concern in both the approaches. Note that the bottom-up parsing can take place from left to right or right to left. Many's the time, a combined

approach is also used.

Transformational Grammar

The transformational grammar comprises tags that are used to handle the plural or the passive sentences. This grammar is an extension to the CFG. So, the parse tree generated by the CFG is further refined with the addition of tags and re-structuring. Commonly, the tree generated by CFG is called *deep structure*, and the one with the transformational grammar is called *surface structure*. The reason behind this is that it consists of the additional tags that enable to understand the context-sensitive entities. Figure 12.3 depicts the structure of transformational grammar.

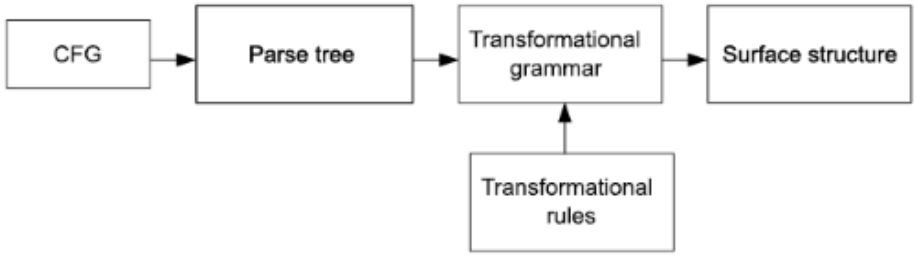

Figure 12.3 Transformational grammar—Tree building.

From Figure 12.3, it is clear that the transformation takes place after the parse tree has been generated. So, what is it doing with the transformational rules? How are they different? The *transformational rules* are the ones that are used to perform the re-arrangement. This is done so as to get valid sentences. The re-arrangement consists of addition or removal of the tokens, or simply re-ordering them. As an example, consider a statement 'Milli is dancing'. The tree structure by CFG identifies the same sentence. But if we change it to 'is Milli dancing?', it is difficult to recognise. Here, the transformation rule is applied for re-ordering. This rule is called *auxiliary inversion*.

Augmented Transition Networks (ATN)

To check the correctness of the sentence grammatically, ATNs are used. Basically, ATNs represent the grammars in NLP and parse them. But prior to its study, an introduction to *recursive transition network (RTN)* is given, which is the core concept for augmented transition networks (ATNs). RTNs are based on the finite state automata, but are equivalent to push down automata. Let us start with RTN.

An RTN is a directed graph. The graph consists of states and arcs that are labelled. It has a start state and the Final state is marked with '/F' or '/1'. What does the RTN do? It takes the input sentence and tells whether the string is acceptable or not. Hence, it performs the syntactic check. The arc indicates the labelled category or can even refer to the other networks.

That is the labelled category can be a start state for another network. A sample RTN is depicted below (Figure 12.4):

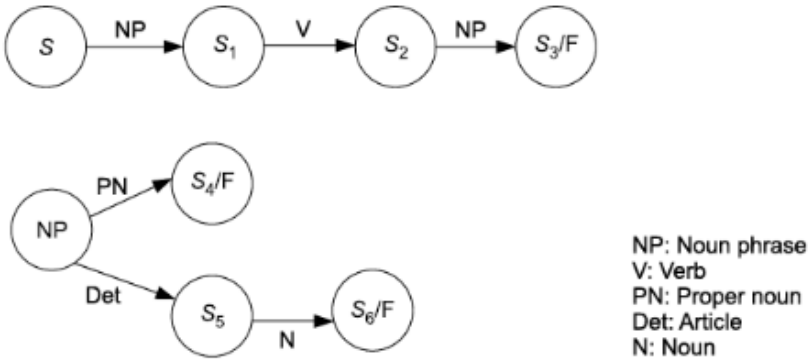

Figure 12.4 Sample RTN.

A RTN has

1. A set of finite terminals, say T.
2. A set of non-terminals.
3. A category C and a mapping $T \rightarrow C$.
4. A start symbol S.

An ATN differs from RTN. In ATN, it is essentially an RTN that has a set of test rules, which need to be satisfied prior to the arc traversal. While doing so, it makes use of registers to save the states. So, in ATN's, we would have a table consisting of arc, test and action. A sample snapshot of a table is given Table 12.1. A sample ATN is shown in Figure 12.5. Do remember that the baseline for ATN is the transformational grammar.

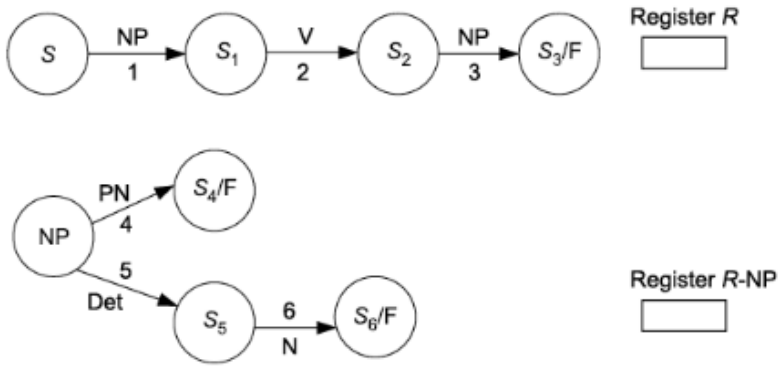

Figure 12.5 Sample ATN.

Let us take a simple example 'The cat drank the milk.'

TABLE 12.1 TABLE LISTING TEST CONDITIONS FOR ARC TRAVERSALS

Arc	*Test*	*Action*
1	NT	Set register to the subject, the network moves to the next network
2	Agrees to subject	Set register with the verb
3	NT	Set register to the subject
4	NT	Set register with the subject

To perform parsing using ATN, for the given sentence,

1. Start with the leftmost node, i.e., *S* for the graph.
2. Let the pointer points to current node/state.
3. Select the arc to traverse such that the associated test from the table is satisfied.
4. In the process, save the registers as required and indicated in test table.
5. If we end up in the terminal node with no inputs being available, the parsing is said to complete.

For the given example, from the first arc itself, another ATN is being referred. So, the test of subject is being registered and the ATN pointer moves to the next network. Once the tests and parsing for this subset are done, i.e., 'The cat' is accepted, it returns back to the previous states and continues to parse till no input is left. At this, the parsing is complete. At any point of time, if there is no arc left for traversal and input text is still remaining, the parsing fails.

A very recent study revealed the use of ATN's for garden path sentences. *Garden path sentences* are the ones in which parsing could result into ambiguity. This ambiguity occurs as humans tend to interpret the sentences, one word at a time. It generally happens when the entire sentence is not heard due to which one cannot predict the meaning. So, it becomes confusing. An example can be 'Because Riya exercises a half hour walk is not difficult for her.' In the initial interpretation, we think of that Riya exercises for half an hour, but actually from 'half hour', a new phrase is beginning. ATNs tend to resolve these misleading garden path sentences too.

So, to summarise, the outcome of syntactic analysis is the ordering structure or the sequence of the words. In short, it determines the structure of the sentence. To make this analysis efficient, grammar plays the most significant role.

12.2.2 Semantic Analysis

With the study of syntactic structure building, i.e., parse tree generation, let us understand the next level of semantic analysis. With semantic, we mean to say the 'meaning'. The meaning of the words that are parsed. Though syntax analysis performs the parsing, it is necessary to understand the meaning of the tokens. So, what are we doing in semantic analysis? (Remember one thing that for each of the tasks NLP is performing, there is some output that you expect.) In semantic analysis, we are having some representation that contain the meaning of the input sentence. How can we have this? Is this again some grammar? Let us understand it better. Humans have prior knowledge about the words to understand their meaning. So, when we talk about word, say table, we have some knowledge about it. Consider two sentences—'Keep the book on the table' and 'Learn the table of number 23'. The contexts in which the table is referred are different. This understanding is done in semantic analysis. How will a machine know that we are referring to physical object table or mathematics concept of table? Let us discuss the different aspects of semantic analysis.

Lexical Processing

In lexical processing, the meaning of the tokens (words in the sentences) is found out. There is a concept of word sense disambiguation here.

Understanding the meaning of a particular word in the context is called *word sense disambiguation*. So, it is concerned with the sense where it would be operational. In order to determine this, additional information about where it would be operational can be attached to the knowledge. For the example of the table, we can have additional information, called *semantic markers* that help in the identification of the meaning. So, the markers here can be physical entity or a mathematical entity. With relation to the verb 'keep', it will be physical entity, whereas with 'learn', it will be mathematical one.

Semantic Grammars

The syntax analysis performs the parsing. Don't you think even though the meaning of a sentence is wrong (like 'The pen is on the ceiling'), it would be parsed? What would happen if the parsing is done where there is no proper meaning to the sentences? This issue is handled by the semantic grammars. So, these grammars perform the semantic as well as syntactic checking tasks. ATNs are used with the semantic grammars.

This grammar has syntactic category that corresponds to the group of words. Then, it is possible to infer the meaning that is generated after the parse.

For example, the grammar can be like

$S \rightarrow$ Action the Food
Action $\rightarrow$ eat|drink|shallow|chew
Food $\rightarrow$ burger|sandwich|coke|pizza

This can generate the meaning so as to perform an action task on a particular food for sentences like 'Eat the pizza' or 'Drink the coke'. Sometimes, the semantic action is written in curly braces after the rules. With this parsing, we are able to get the meaning as well. But what can you say about the rules? They would be large enough, as we are not taking into account the generalisations!

Case Grammar

Case grammar is also called as *Fillmore grammar*. The grammar introduced by Fillmore in 1968 helps in characterising the relationships between the verbs and the nouns. Here, the rules are formed that discuss the syntax instead of semantics.

Heads or the leads are defined for the verbs. These leads are linked with some particular role that they play. These roles are the cases. Some of the elements of the set cases are given below:

1. Object (thing on which it is acted)
2. Agent/actor (someone who carries out the action or the event)
3. Dative (someone/something who is affected by the event)
4. Instrument (cause of the event)
5. Location (place where the event/action occurs)
6. Time (date or time at which the action/event takes place)

So, each verb sense accepts finite cases. Some cases are compulsory, while some are optional. Let us take an example, say 'Rohit will meet Kunal at mall'.

The event here is of 'meet'. So, this becomes the head. The details of set cases are depicted in Figure 12.6 for this head.

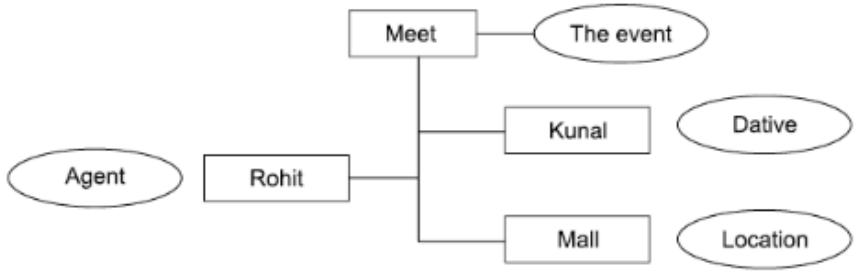

Figure 12.6 An example for case grammar.

As discussed earlier, in some events, the cases could be optional. For example, an event like 'hit'. There can be a sentence like 'Shyam hit the ball' or 'Shyam hit the ball with bat'. So, for 'hit', we can have agent, object, and (instrument).

The underlying notion for the case grammars is the *case frame generation*. These are also called *case slots*. For first sentence, the case frame is shown below:

[<hit>
[Agent: Shyam
Object: ball]
]

One major advantage of using case grammars is their capacity to handle noisy or incomplete sentences. For example, 'Ram ran'. It is possible to represent this type of sentence with case grammar. The only issue occurs when there is a need to tackle complicated syntax. ATNs are also used for building appropriate structures to carry out parsing.

Conceptual Dependency

This is a parsing technique that explicates the meaning. Conceptual dependency (CD) makes case frame representations for the verbs that are common using conceptual dependency structures. So, small primitive actions are formed. These primitives, called *ACTS* comprise a fixed set of cases. As discussed by Shank, some of the primitives ACTS are given below:

1. *ATRANS*: It represents the transfer of some relationship for verb, say 'give'.
2. *PTRANS*: It represents the transfer of location of an object, say verb 'go'.
3. *GRASP*: This primitive occurs with verbs like 'throw'.
4. SPEAK: This primitive has some sound production, say the verb 'speak' or 'say'.

The sentence—'Kavita went for a movie' falls under the PTRANS. The CD structure

for the verb is given in Figure 12.7 in which PP indicates the picture producer. In short, it is an entity that can be concrete or abstract. The bi-directional arrow indicates the conceptualisation between the ACT and Kavita. There are other concepts too like picture aider (adjective) and action aider(adverb) referred to as *PA* and *AA*, respectively. The PP, ACT, PA and AA are the conceptual categories and their relations are the dependencies.

Figure 12.7 A conceptual dependency structure.

While generating a parser that employs the conceptual dependency, the most important part is the ACT primitive formulation. Same verb can exist in different environments. For example, a verb like 'want' can have different ACT. Some examples of the parsers that use CD are SAM, MARGIE.

To conclude, for the semantic parsing, it is very much essential to have an appropriate structure representation so that the meaning of the sentence is determined.

12.3 DISCOURSE AND PRAGMATIC PROCESSING

We now move towards the next level of discourse and pragmatic processing. After performing the semantic analysis, where the understanding of the sentence is clear, it is now necessary to understand and recognise the individual specific tokens. For example, a token like I could be some individual, say Danny. This involves discourse integration. In pragmatics, the context and what the user wants, needs to be analysed. So, it is more concerned with the speech acts or the conversations. This is done with the proper knowledge base. For more clarity, we can say that pragmatics and discourse analysis is concerned with the study of the meanings of the words, where they can be explained using some knowledge with reference to the context and the environment (oral or written) in which they occur. So, in short for discourse and pragmatic processing, the context is very much important and so are the relationships between the sentences. Do remember discourse analysis is often referred to as a *sub-field of pragmatics*. There are a number of relationships that exist, a few of which are mentioned below:

1. *Anaphoria:* Consider an example—Sam had a blue car. It was a big one. Here, it is understood that the word 'it' is used for car.

2. *Part of objects/entities:* Consider an example—Sam opened the door of the car. He found that the CD player was stolen. Here, the relation of understanding the CD-player as a part of the car is considered.

3. *Names of the objects/persons:* Consider a sentence—Sam went for a drive. Here, Sam as a name is identified.

4. *Planning sequences:* Consider the sentences—Hari wanted to get a mango from the tree. He got a big stick. The reason behind getting the stick was to reach the mango.

These are just a few relationships that need to be recognised. How are we going to do that? Still things are not clear what we are doing in pragmatics and discourse analysis. Let us go back to the first dialogue mentioned in the introduction of this chapter. You have inferred something from the dialogue. This inference is concerned with the pragmatic and discourse analysis. If you hear a sentence, the understanding of that sentence involves answering to questions—who said that? To whom it was addressed? Where was it said? What was it all about? and so on. So, with this, you infer. Naturally one has the knowledge and capacity to understand. Coming back to the machine level, a knowledge base is required and that too strong enough to have this understanding of the relationships as discussed when provided with the same text. We do not go into the representation part for the knowledge bases here. Let us look at what knowledge is to be used in this analysis.

1. *Dialogue focus:* This is concerned with understanding the relevance, i.e., the part which is relevant to the knowledge base. This knowledge needs to be taken care of along with ambiguity handling.

2. *Modelling the beliefs:* It involves expressing or representing the belief while participating in a dialogue. If you have a dialogue with somebody, it is necessary that you represent your belief as well as belief of the other participant too. Mapping it to the machine world, the program with which (let us assume) the communication is taking place, there is a need to have proper representation of the program's belief and the communicating person's belief too.

3. *Goal planning:* In some cases, to understand what the text is all about, we need to understand the overall meaning. As said earlier, the meaning in terms of dialogue or the beliefs is important but goal planning is more about goal and plan recognition. The example of planning sequence relationship where the goal of Hari was to get a mango and he planned to get a stick for it comes under this knowledge usage. Though the goal here was getting the mango, there can be different goals. There can be career goals like getting admission to some renowned school or achieving certain position. These would fall under the category of achievement goals.

4. *Speech acts:* What we say verbally could be an advice, a threat, a warning, a request and so on. These all come under the speech acts. A simple sentence—'Read the chapter till the end' is a speech act. Speech acts fall into the different categories mentioned above. These are modelled in a different way for understanding and play an important role in

pragmatic analysis. Recent study shows the use of speech acts for message board posts, where typically, in forum, a question-answer session takes place and the categories are uniquely identified.

So, we have discussed about the different relationships and their contexts based on the knowledge that build the pragmatics and discourse analysis. But is it the only way pragmatics and discourse analysis work? Can it involve focusing on other aspects rather than just words like—gestures, expressions and so on? Yes, indeed, and this is referred to as *multi-modal discourse analysis*.

12.4 INFORMATION RETRIEVAL

Let us first understand the problem of information retrieval. So, what is it? Suppose you want to retrieve information about first players who scored 200+ runs in a single inning in ODI. So, you are in discussion with your friend who knows about cricket. First, you will get information about all players who scored 200+ runs, and then you will get years in which they scored. This simple retrieval will, conclude that Sachin Tendulkar is the first player to score 200+ runs. Similarly, we come across various scenarios in everyday life which demand information retrieval.

A very common example that we have discussed in the introduction is of the search engines. The search engine performs information retrieval (IR). We type in a query and the information required by us is retrieved. Figure 12.8 depicts the basic IR system.

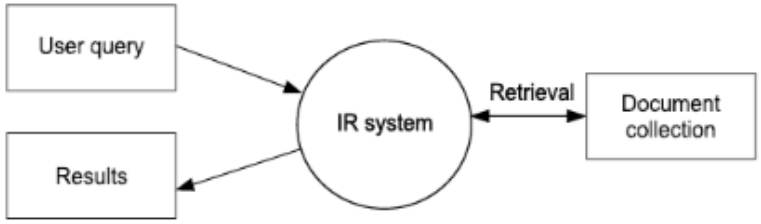

Figure 12.8 An IR system—Getting relevant documents.

We have talked about the search, but the question here is—is it possible to have an optimal way that can be exploited by the search engines for retrieval? What could it be and how would the things work here?

12.4.1 IR Models

We begin with the different models. These models are the mappings for the data that are used to retrieve the information. Let us discuss them.

Boolean Model

In Boolean model, the tokens are treated in the form of 1's and 0's. The presence or absence of a term is marked accordingly. So, the document is represented in the form of $t_1 \wedge t_2 \ldots t_n$, where t_i forms a term. The query can be $t_1 \wedge \sim t_2$. The Boolean mapping is found to be the simplest one that was used in earlier systems. With the Boolean model, the requirement is to find the probability of relevance R for a document d and a query q. It is represented as follows:

$P(R|d, q)$, where R can be true or false.

But it has a lot of drawbacks. Since it is dependent on the bit presence, the ordering becomes an issue. (In what way the relevant ones are to be presented?). Further, matching similar words, for example, if the terms are cardigan and sweater, then it becomes another issue. Most importantly, a question raises up here—is it possible as an end user to formulate the Boolean query? Though it is pretty complex, still, many probabilistic approaches have been developed with Boolean models.

Bag of Words

The frequency of a word is taken into account instead of ordering. Consider the following texts:

Rohan drives Audi. (text 1)

Sameer drives BMW. He drives Mercedes also. (text 2)

(At a very basic level, to understand the concept, the words identified here are Rohan, drives, Audi, Sameer, he, Mercedes, also). The dictionary or the bag of words has these tokens. So, a vector for the text 1 and 2 would be

[1, 1, 1, 0, 0, 0, 0] for text 1

[0, 2, 0, 1, 1, 1, 1] for text 2

The count in the vectors is the frequency of occurrence of the identified terms in the text.

Vector Space Model

In vector space model, the documents and the query are represented as *vectors*. A vector comprises dimensions that are the terms used to index it. The model is given below:

Vector comprising the terms $<t_1, t_2 \ldots, t_n>$

The document and query are formed as $<p_1, p_2, p_3 \ldots p_n>$, where p_x represents the weight of the term x in the document. The query is represented as $<q_1, q_2, q_3, \ldots, q_n>$.

Relevance of a document d to the query q is computed based on the similarity between them. There are various functions to do so. The different ways to compute the similarity measures are cosine, dot product, dice, Jaccard and so on.

Let us proceed with the matrix representation in vector space model. Figure 12.9 shows the matrix.

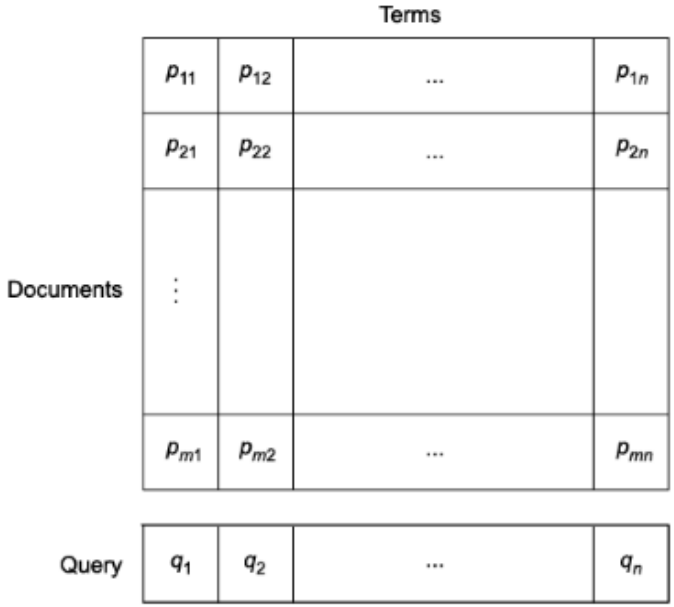

Figure 12.9 Matrix representation.

Term-frequency, document frequency and inverse document frequency: The concept of term frequency and inverse document frequency is highlighted now. Most often what is observed is when we refer to the term, the weights are the occurrence of the terms in the matrix. So, the *term frequency* is defined as the frequency of occurrence of the term in the document. It is represented as *tf*. The document frequency tells about the number of documents that have a specific term. It is represented as *df*. Inverse document frequency determines the unevenness of the term in the distribution throughout the documents. This is represented as *idf*. The weights are now computed as follows:

P_{11} = Weight of 1st term in document 1

So, to make it more generalised, we have (Remember the term frequencies are normalised by maximum frequency count of that term.)

Weight of a term in $d = wt(t, d) = tf(t, d) * idf(t)$

where $tf(t, d)$ is the occurrence of the term t in d.

$idf(t)$ = log (Total number of documents/Documents containing t term)

Probabilistic Model

Bayesian probability is the most common mechanism that is used in probabilistic inferring. The major drawback of using the Bayesian approach is that it requires prior knowledge. But it is able to address the uncertainty factor that could occur while submitting the query. (The details of Bayesian approach are covered in Chapter 7.)

12.4.2 Pre-processing

Let us start with pre-processing tasks. Why is this pre-processing required? Are all the terms equally important during retrieval? There are terms like 'a', 'and', 'the', 'to', etc., which occur quiet frequently in the document. These terms may not be equally important as some other terms in the document; hence, to have an appropriate representation, pre-processing is essential. Some of the pre-processing tasks are as follows:

1. ***Tokenisation:*** It refers to separation of the tokens from the text. In simple words, it is concerned with the removal of punctuation marks or some characters that are not required. For a sentence—'Thanks for treat!', it would return 'thanks', 'for' and 'treat' as the tokens.

2. ***Stemming:*** In stemming, the keywords are used as they are or they can be transformed. For example, we can have words like 'playing', 'played', 'plays' which together can become 'play'. This is carried out in stemming. So, stemming reduces the distinct terms that are used to refer a word. *Porter's algorithm* is the most common one in stemming. This results in a better performance for retrieval.

3. ***Lemmatisation:*** Here, the base form of the word is returned. For example, in case of 'heard', it also returns 'hear'.

4. ***Case folding:*** This is simply conversion of the case. If a word like 'PLAY' occurs, the case folding converts it into 'play'.

5. ***Stop word removal:*** The words like 'the', 'on', 'after', 'an', 'it' and so on are called *stop words*. It is of limited use to assign weight to them in the retrieval part. Thus, the removal of the articles, conjunctions, prepositions is carried out in stop word removal.

6. ***Normalisation:*** Many times, this is also called *equivalence classing*. Here, equivalence classes are built to have correct mapping, where synonyms too could be considered.

Generally, spell checking option is also performed. (You must have noticed this while using search engine.)

12.4.3 Indexing

Remember the indexing that you have studied in the data structures. *Indexing* is a technique in which the index terms and their related information essential for retrieval are

stored. So, what is indexed? The document corpus is indexed. What is related information? This related information consists of the term weights and their postings. What are these postings? The *posting list* is the document number that contains the term.

The concept is expressed with *inverted file index*. This is used to have efficient retrieval. In indexing, the following steps occur:

1. Tokenising of terms from the document corpus.
2. Performing other pre-processing tasks (where the stop word removal, case folding, etc. occur)
3. Indexing the documents with the creation of inverted index file (This file consists of dictionary and posting list. A sample snapshot of the same is given in Figure 12.10).

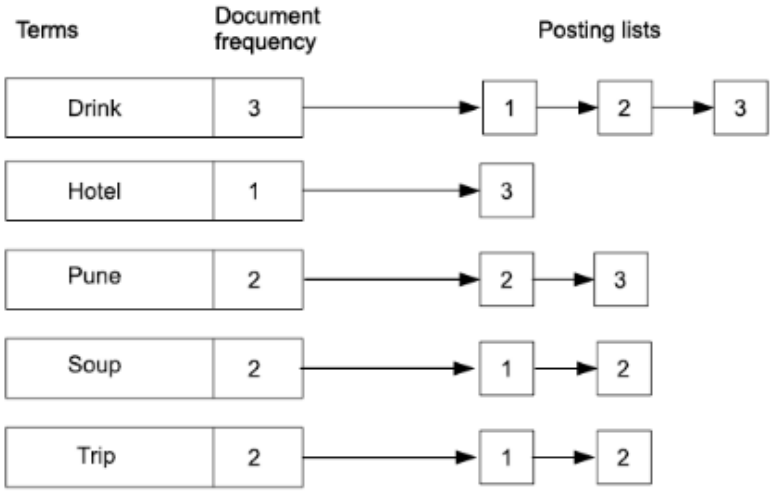

Figure 12.10 An index structure.

The left-hand side of Figure 12.10 shows the terms that are in sorted order, whereas the right-hand side has the posting lists. It might store some more information that is essential to make the retrieval better. The document along with the weight for the term (not shown in Figure 12.10) can also be maintained in the posting list. This information proves to be useful when ranking comes into picture.

Getting Right Answers

On what basis, can we say the IR system is giving us appropriate results or the ones that we actually expect? There are a few computational formulae to determine them, which are mentioned below:

1. ***Precision:*** It refers to measuring the proportion of retrieved documents that are actually relevant. It is given as follows:

Precision = Number of retrieved relevant documents/Total number of relevant documents

Note that precision is actually nearer to accuracy.

2. ***Recall:*** It refers to measuring the proportion of relevant documents that have been retrieved. It is given as follows:

Recall = Number of retrieved relevant documents/Total number of retrieved documents

Also, F-measure is used that combines the recall and precision. It is given as follows:

F-measure = (2 * Precision * Recall)/(Precision + Recall)

3. ***Confusion matrix:*** It is a matrix that is often used in classification purposes and tells us about the accurate prediction of the classes. Coming back to the IR system, Figure 12.11 depicts a confusion matrix, showing us whether the retrieved documents are relevant or not. We can treat them as two classes and identify the misleading one. *True positives* are the relevant documents that are correctly retrieved, whereas *false positives* are non-relevant documents retrieved. Similarly, *false negatives* are the non-retrieved documents that are relevant, whereas *true negatives* are the non-relevant documents that are not retrieved.

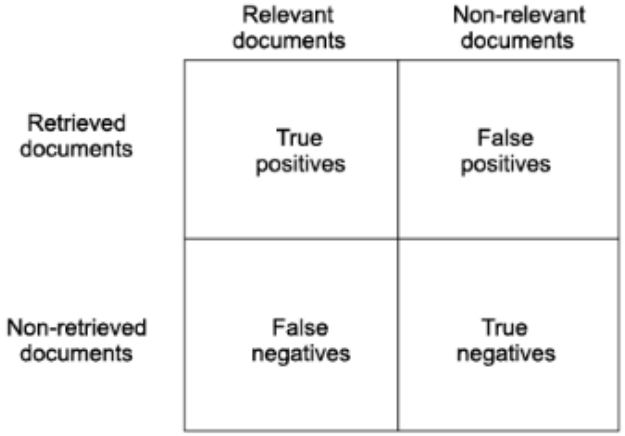

Figure 12.11 Confusion matrix.

12.4.4 Similarity Measures

For different text documents or for computing the similarity between the two texts, we discuss different similarity measures. Let us discuss the cosine measure here. A *cosine similarity* measures the angle between the two vectors. In cosine similarity, if the value is zero, then there is no similarity, whereas with 1, they are similar. Let us say we have formed two vectors v_1 and v_2 for two sentences—'Read AI book' and 'Read AI to understand AI'. Let us use bag of words approach for vector formulation. So, the tokens are 'read', AI, 'book' and 'understand'. The vectors would be

$$v_1 = (1, 1, 1, 0) \text{ and } v_2 = (1, 2, 0, 1)$$

The cosine similarity is given by the following formula:

$$\cos \theta = \frac{v_1 . v_2}{\|v_1\| \|v_2\|}$$

So, numerator = 1 * 1 + 1 * 2 + 1 * 0 + 0 * 1 = 1 + 2 + 0 + 0 = 3

and denominator = $\sqrt{1^2 + 1^2 + 1^2 + 0^2}\ \sqrt{1^2 + 2^2 + 0^2 + 1^2}$ = 1.73 * 2.44 = 4.2212

$$\cos \theta = 3/4.2212 \approx 0.71.$$

What would be the cosine similarity between square shape and rectangular shape? Do you think they will give an exact match? Will things differ if used with *tf-idf*? Try to solve.

To summarise, IR is all about getting the required relevant information. With the various techniques in practice to find the similarity between the inputted query and the documents, search engines put to use ranking techniques too additionally. There could be clustering of relevant results, where similar sets are grouped in one group (k-means and other techniques) for providing the user with relevant data. At the back-end, prior classification too can take place. Further, depending on the user's response to the ranked results, a feedback mechanism too can be used, where the needs of the user are best served. This further improves the ranking and makes the IR system an efficient one as well as personalised one.

12.5 INFORMATION EXTRACTION

In information retrieval, we have studied about getting the information—the search engines being the common example. Now, what does information extraction (IE) do? Is it not the case that information extraction and retrieval are the same? Do remember that in information retrieval, the data is not at all modified or changed. Simply, the relevant data is presented to the user. In IE, template matching is carried out. The IE module could make entries in the databases. So, there is a pre-defined fixed format in which the text entries are carried out. In short, IE makes things structured from unstructured inputs. With IE, the information from the documents is extracted into the templates. Are you feeling like there is some similarity between retrieval and extraction? IR gets relevant documents, whereas IE gets relevant information from the documents. Though there could be an overlap between them. IE could lie between IR and text understanding. Figure 12.12 depicts the basic IE system.

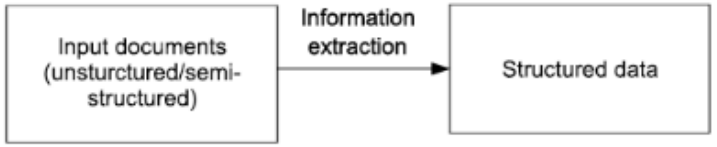

Figure 12.12 Information extraction.

Let us take an example to explain IE. An employment agency has to send mails to the clients whose profile suit some ABC company's requirements. From the CVs of these clients, only the mail-id needs to be extracted to send the date for interview. In another example, a company wants to categorise candidates based on their skill set. In this case, the information about the skills is retrieved from their CVs. Here, terms like Java, .Net etc. are used for template matching. This is also a task of IE.

There are various aspects of IE such as what is to be extracted?, what is the input data? what methods are to be used?, How is the output expected? Let us try to address them. To extract, it could be any particular entity, or any specific table or relations. Whereas, the input data could be a simple set of documents, web pages and so on, which is unstructured. The methods applied can be handwritten patterns, learning-based methods, rule-based methods or statistical methods. Figure 12.13 shows the basic modules of IE.

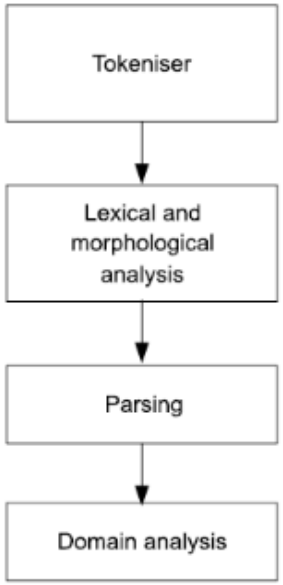

Figure 12.13 Modules of IE.

What is the job of each of them? Tokeniser, as we are already familiar, does the word separation. In *lexical and morphological analysis*, part of speech tagging is done. Identifying the parts of speech or learning the word sense is carried out here. In *parsing*, syntax analysis is done, whereas in *domain analysis*, merging the partial results or co-reference is carried out.

FRUMP, LSP were the early IE systems. Later on, systems for web-based information extraction were Crunch, Content Seeker, etc. At present, wrappers are used on a large scale. They are called *information extraction procedures* that extract some content or bits from the text. Meta-crawlers are the type of wrappers. Currently, the focus of the research is on investigating and inventing different techniques for text summarisation as well as on cross-media and language-based IE systems.

12.6 MACHINE TRANSLATION

It is the process of translating source language into target language. This has to be necessarily natural language. Why do we need this translation? Generally, you refer to a number of sites to get some sort of information. Let us say you want to have some paper related to AI, you get a link for downloading it and you find that the paper is presented by a Spanish author in Spanish! You need machine translation (MT) here. Some of the aspects it needs to know are the target language, and the source language, to have a detailed knowledge about the contents, to understand the 'context' in which things are written or said (and speech analysis as well). Figure 12.14 below depicts the basic steps required for MT. As it can be observed from Figure 12.14, there is need to perform the syntax and semantic tasks too. Then, the representations from the source to target also exist. The representation is referred to as *interlingua*. Overall in MT, three major techniques exist for this translation, one is direct (morphological one), other is transfer (syntax and semantic), and the last one being the interlingua. Interlingua is the most difficult one and makes the analysis work more, thus increasing the cost. In interlingua, the dependencies and the language-specific distinctions are accounted. So, it is a single representation for the source and the target language. Today, MT systems are also looked upon as research areas. But it is very difficult to have MT systems developed. Can you think of the drawbacks that would arise in building these systems? In some source language, there is a single word that tells something. For the same thing, in another language, there could be 4–5 words that are required. Further, the meanings and mapping of the similar words add up complexity. The parts of speech are different. Hence, developing MT systems has become a challenging task.

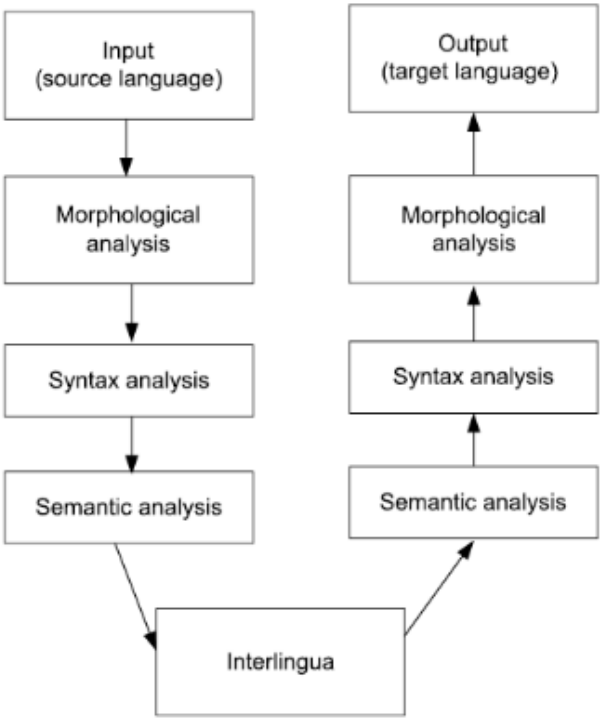

Figure 12.14 Steps of machine translation.

12.7 NLP AND ITS APPLICATION

From the study of the entire concepts regarding NLP, we now move towards the applications of NLP. Let us look at one of them to understand when, where and in what way these concepts are used. In the sections discussed earlier, we have mentioned about IR with search engines. NLP techniques can enhance them to have a personalised search. Suppose a person enters 'apple' as a query to search engine. He tends to get information about Apple products, and the expected result will also include a information regarding fruit apple. So, he gets information about the fruit apple too, but often these web pages have lower ranks. What we mean to put forth here is that the results expected are dependent on the profile of the user, or it even could be decided on the basis of previously clicked/accessed data. So, in turn, this historic data of the user can form and contain potential information that can be utilised for helping the user in getting personalised results.

How can this historic data be useful? With the NLP models, weighing schemes can be put to work to have similarity factors computed. For this, search engines use previously accessed results. These weights can be refined by re-assignment on the basis of statistical methods that inform about the accessibility pattern. Each term/word captured and accessed, thus, has increased weight. Further, word and concept ranking can be employed to have re-ranking schemes. This approach can be employed with different models to have the representation of the data/text. Moreover, a feedback mechanism can add up to assist the overall personalisation techniques.

SUMMARY

This chapter discusses about NLP along with preliminaries that are needed to be understood. NLP techniques have gained a lot of importance recently, showing larger prospects for research. They are important in the perspective to understand and infer things. With the various techniques discussed in syntax and semantic analysis, today there is a need to appropriately represent the content and the context. The use of ATNs has been proved effective in the analysis of natural language with the required representation structure. The grammar and knowledge base too play a vital role in processing. With CFG and Fillmore grammar, the rule mapping can be carried out with ease. Parsing is another important task that helps in getting the syntax correct, whereas pragmatics and discourse analysis handle speech.

Further, with the wide use of search engines like Google, information retrieval has taken up a big leap. New techniques are in development phase, where efficient retrieved and ranked results are made available to the user. Information extraction too is an important aspect of handling NLP. There are learning and rule-based systems that help in the information extraction. With the increasing information, getting the precise data in a structured format is now looked at. Machine translation, yet other aspect of NLP, helps in the conversion of source languages into target language, and handles speech too.

All the steps or the topics discussed in NLP are closely related to each other and one would find an overlap among them. NLP can also be used in many applications like sentiment analysis, product recommendation system, etc. To conclude, we can say at present, NLP finds a noteworthy position in the area of research and if one is looking forward to it, he should not forget the text and the context.

KEYWORDS

1. **Natural language:** It is a language that is in use and has been developed naturally. It is in practice and is a way for communication that can be in oral or