Chapter 15
Automated Planning

Abstract Automated planning deals with the tasks of finding out ordered sets of actions that allow a system to transform an initial state to a state satisfying goal specification. The set of actions is a plan, and it belongs to PSPACE-complete. Automatic planing or scheduling generates a set of actions automatically. This chapter presents the nature of automated planning, the classical planning problem, agent types that execute the problem, and worked examples. It also covers, the concepts and implementation aspects of forward planning, partial-order planning, planning languages, a case study of general planning language—STRIPS, and search strategies. Planning with propositional logic, planning graphs, and hierarchical network planning are demonstrated. The multiagent planning techniques are presented for goal and task refinement, decentralized planning, and on how to do coordination after it is planned. This is followed by the chapter summary, and a set of exercises.

Keywords Automated planning · Scheduling · Forward planning · Partial-order planning · Planning languages · STRIPS · Multiagent planning

15.1 Introduction

Automated planning is concerned with the problem of finding a set of actions to be carried out in an ordered way to complete a job, such that they allow a system to transform an initial state into a state that satisfies the goal specification. In a deterministic system, these sets of actions are called *plans*, while in nondeterministic systems they are called *policies*. Finding a plan, or even deciding its existence, has shown to be *PSPACE-complete*[1] in the deterministic planning, unless severe restrictions are applied to the search [1].

Developing automated methods for reasoning about plans and schedules and for generating the plans has remained the part AI, which helped both the autonomous

[1] In the theory of computational complexity, a decision problem is in complexity class *PSPACE-complete* if it can be solved using a memory whose size is polynomial on the size of input (i.e., *polynomial space*), and if every other problem that can be solved in polynomial space can be transformed into it in polynomial time.

K. R. Chowdhary, *Fundamentals of Artificial Intelligence*,
https://doi.org/10.1007/978-81-322-3972-7_15

and human agents. There is always a need for planning when an agent wants to control the evolution of its environment. An algorithm for a planning problem has the following inputs:

1. possible courses of actions,
2. a predictive model that underlies the dynamics, and
3. a performance measure that evaluates the courses of actions.

The output or solution to a planning problem is one or more courses of actions that satisfies the specified requirements for the minimum performance. Hence, a planning problem involves deciding "what" actions to be performed and "when" they should be performed. The "when" part of the problem has been traditionally called the *scheduling problem*. The separation between "what" and "when" is motivated by computational considerations. In fact the algorithm for automating scheduling has been around for a long time, for example, in the areas of operations research as well for automating the complete planning problems.

The classical scheduling methods are concerned with the planning problems where resource constraints are static, and prespecified in numerical form. This is in contrast to the AI methods for scheduling that allow declarative specifications for both symbolic and numeric resource constraints and handle dynamic changes in the constraints, i.e., while in execution. The Constraint Satisfaction Problems (CSFs) (see Chap. 10) form the canonical backbone of most AI scheduling methods, and there are several sophisticated heuristic search strategies that are common for such problems [11].

Although the classical panning model has been prevailing for a long time, a majority of research in planning is toward planning in environments that are *dynamic*, *stochastic*, and in a *partially observable* world. These requirements are not compatible with the classic planning assumptions, which are for deterministic, static, and fully observable world. To cope with the partially observable environments, gathering of information is designed as part of the planning activity, and the existing classical planning techniques are extended to accommodate interleaving of scheduling and planing tasks. The environments that are stochastic are modeled using Markov decision processes, while the planning in such environments requires construction of policies for the corresponding Markov decision processes [6].

All variants of domain-independent planning problem are known to be computationally hard (*P-SPACE complete* or worst); efficiency can be improved only through exploitation of problem distribution space and through the knowledge of domain structure. For improving the efficiency of plan generation, formal planning models are preferred. Following are the basic approaches for achieving efficiency enhancement:

1. split both, domain and the problem into tractable sub-components using decomposition techniques,
2. find out the control information using inductive and speedup learning techniques,
3. define abstractions for expected problem distribution, and
4. define language(s) to express domain-specific search control information.

Another approach to planning is extending and complementing the planning models that are based exclusively on techniques that define subgoals. A promising approach is using constraint-logic-programming-type models that allow both: defining subgoals and constraint-satisfaction criteria. These work together in controlling the planner, which results in handling action selection and scheduling of planning problems through appropriate computational models. In addition to the above, plan generation can also be carried out using network flow approaches, which are based on a form of disjunctive projection of future states of the agent [9].

So far, both the *classical* and *stochastic* planning techniques are in common used in the design of autonomous agents. The AI planning techniques are also common in software domains such as database query planning and Internet browsing, however, they have less impacted the problems that support industrial applications. Although the industrial area has an enormous scope for planning problems—be it project planning, process planning, or maintenance planning—the automation so far has progressed to supporting scheduling, particularly concentrating on harder *action-selection* problem, which are more concerned with the humans domain.

The type of plannings, which are *incremental* or *interactive* planning in nature, are new areas in plannings. However, many planning applications involve reasoning with a variety of constraints that are not of temporal type, e.g., the problems of path planning, assembly planning, and manufacturing problems need *spatial* and *geometric-reasoning* capabilities. Due to the nature of these constraints, it is neither possible to ignore or abstract away, nor it is advisable to encode them in a homogeneous representation. These conditions make it necessary to have an interaction between the planner and the reasoners. Hence, understanding of the modalities of interaction between a planner and the external reasoners, which may be both humans or machines, is thus important [7].

Learning Outcomes of This Chapter:

1. Define the concept of a planning system and how it differs from classical search techniques. [Familiarity]
2. Describe the differences between planning as search, operator-based planning, and propositional planning, providing examples of domains where each is most applicable. [Familiarity]

15.2 Automated Planning

Automated planning techniques are now commonly being applied in a number of tasks, that include, robotics, process planning, web-based information gathering, autonomous agents, and spacecraft mission control. A solution to a problem in automated planning can be described in terms of a sequence of steps that transforms some initial description of the problem state, for example, the initial configuration of a puzzle, into a description satisfying a specified goal criterion. The steps (transformations) are called *operators* and a problem is defined in terms of a set of operators

and a language for describing problem states. The problem states correspond to instantaneous descriptions of the world and operators correspond to *actions* that an agent can perform to change the state of the world [3].

The planning is about how an agent achieves its goals, even the simplest ones an agent must reason about the future. Since the goal is not achievable in a single step, the number of steps to be carried out needs to be broken up into subtasks, and steps for each needs to be the goal; what an agent will do in the next step also depends on its past.

To complete each subtask, there are actions, which need to be carried out; each action has a subsequent state as well as a preceding state. To be simple at start of this subject, the following assumptions are made:

1. the actions are deterministic, i.e., the agent can determine the consequent of the actions,
2. the world is fully observable, i.e., the agent can observe the correct state of the world, and
3. the closed world assumption, i.e., the facts not described in the world are false.

15.3 The Basic Planning Problem

A basic planning problem usually comprises an initial world description, a description of the goal world, and a set of actions (sometimes also called *operators*) that map a world description to another. A solution is a sequence of actions leading from the initial world description to the goal world description, referred to as a *plan*.

A *deterministic action* is a *partial function* from states to states.[2] For example, a robot can move block x onto y, if x has top-clear, y has top-clear, and obviously, $x \neq y$. Also, all the alternatives are not available. A *precondition* of an action decides about when the action can be carried out, and whether the resulting state due to an action is the effect of the actions.

Definition 15.1 (*Planning Problem*) The relevant part of the world is in a certain state, but managers or directors would like it to be in another state. The (abstract) problem of how one should get from the current state of the world through a sequence of actions to the desired goal state is a planning problem. □

AI planning techniques are techniques to search for a plan: *forward planning* is a planning technique building a plan starting from the initial state; *backward planning* starts from the goal states; and *least-commitment planning* constructs plans by adding actions in a non-sequential order.

Ideally, to solve such planning problems, we would like to have a general planning-problem solver. However, such an algorithm, solving all planning problems, can be proven to be non-existing (that is, the general planning problem is undecidable). We

[2]Partial function: Every state "(state, action)" pair does not necessarily result in a state.

therefore, try to concentrate on a simplification of the general planning problem called *the classical planning problem*. Although not all realistic problems can be modeled as a classical planning problem, they can help to solve more complex problems.

15.3.1 The Classical Planning Problem

Classical planning problem is the simplest case of planning, where the environment is static and deterministic, and the planner has complete knowledge about the current state of the world. Two important issues are faced by the classical planners: 1. To model the actions and changes, and 2. To organize search for plans (i.e., sequences of actions) that are capable of achieving the goals. The *logic programming* and *nonmonotonic reasoning* are frequently used in planning and search, however, many implemented planners have used a variant of action model, called the STanford Research Institute Problem Solver (STRIPS). This model represents the state of the world in terms of state variables and their values, and actions as state-transforming functions, which are deterministic in nature. Most of the early planners modeled the state-space exploration of the world-states as a search, and the transitions between states represented the actions.

The state exploration method worked suitable only for problems having small search space, due to the complexity in space and time. Instead, the utility of manipulating partial plans during search became popular, and this led to the design of algorithms that search in the space of partial plans.

The classical planning problem is defined as follows. Given

1. a description of the known part of the *initial state* of the world (in a formal language, usually propositional logic) denoted by **I**,
2. a description of the *goal* (i.e., a set of goal states), denoted by **G**, and
3. a description of the possible *atomic actions* (**R** (i.e., rule)) that can be performed, modeled as state transformation functions,

determine a plan, i.e., a sequence of actions that transforms each of the states fitting the initial configuration of the world into one of the goal states.Thus, classical planning problem is a tuple $\langle I, G, R \rangle$. Consider the following example of planning the "Transport a passenger by cab."

Example 15.1 Classical planning problem of "Transporting by cab."

Suppose that initially (i.e., in all states of the world that match the description **I**), there is a cab at a location A, represented by a binary state variable $cab(A)$, and a passenger at a location B, represented by $passgr(B)$. In each of the states described by **G** the passenger should be at a location C, denoted by $passgr(C)$. Furthermore, suppose that there are three actions (move, load, unload) that can transform (some part of) the state of the world.

Table 15.1 Classical planning problem

State	Transition	Comment
0.	I	; Initial state
1.	$move(A, B)$	; cab moves from location A to B
2.	$load(passgr)$	; passenger gets into cab
3.	$move(B, C)$	; cab moves from location B to C
4.	$unload(passgr)$	; passenger unloads from cab
5.	G	; goal: passenger at location C

Following are the steps for actions:

1. The cab can move from one location to another: $move(x, y)$ with $x, y \in \{A, B, C\}$. This action requires that a priori $cab(x)$ holds, and ensures that in the resulting state $\neg cab(x)$ and $cab(y)$ hold, that is cab is not at place x as well not at the place y.
2. The passenger can get into the cab: $load(passgr)$. This action requires a priori $cab(x)$ and $passgr(y)$ and $x = y$, and in the resulting state both $\neg passgr(y)$ and $passgr(cab)$ (i.e., passenger in cab) should hold.
3. The passenger can get out of the cab: $unload()$. This action requires that cab is at location x ($cab(x)$) and passenger in cab ($passgr(cab)$), and results in $\neg passgr(cab)$ and $passgr(x)$.

With **I** as the initial set of states, and **G** as a set of goal states, the sequence of state transitions can be indicated as shown in Table 15.1. □

15.3.2 Agent Types

Agents are classified according to the techniques they employ in their decision making:

1. *Reactive agents*: They base their next decision solely on their current sensory input.
2. *Planning agents*: They base their course of action considering the anticipated future situations, possibly as a result of their own actions.

Whether an agent should plan or it should be reactive, depends on the particular situation it finds itself in. Consider the case where an agent has to plan a route from one place to another. A *reactive* agent might use a compass to plot its course, whereas a *planning agent* would consult a map. Clearly, the planning agent will come up with the shortest route in most cases, as it will not be confronted with uncrossable rivers and one-way hills. On the other hand, there are also situations where a reactive agent can be at least as effective, for instance, if there are no maps to consult such as in

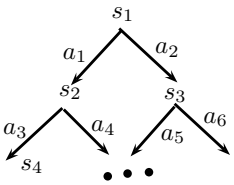

Fig. 15.1 World states and agent (robot) actions

Table 15.2 Table for mappings: State × Action → State

State	Action	Resulting state
s_1	a_1	s_2
s_1	a_2	s_3
s_2	a_3	s_4
s_2	a_4	s_5
…	…	…

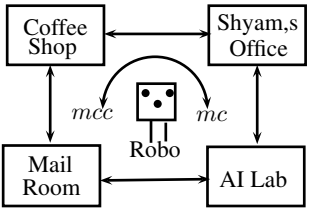

Fig. 15.2 Delivery robot with service locations

a domain of planetary exploration, like Mars or Moon. Nevertheless, the ability to plan ahead is invaluable in many domains.

Considering the *states set* as $s_1, s_2, \ldots,$ and set of *actions* as $a_1, a_2, \ldots,$ they can be represented using a tree shown in Fig. 15.1 or explicitly by Table 15.2.

We consider the following example to better understand the actions and states world, where agent is a robot, called *Robo*.

Example 15.2 A delivery Robot's (Robo's) planning.

A delivery robot shown in Fig. 15.2 is responsible for some jobs. It can pick up mail from the mail room and deliver it to Shyam's office, and also can pick up coffee from the coffee room and can deliver it to Shyam's office. The robot, called *Robo*, can also reach these places by moving clockwise (*mc*) as well as moving counterclockwise (*mcc*). Assume that the mail it handles is a postal mail (not the email). The domain of the world is represented by the terminology described as follows [13].

Various *locations* are represented by the following symbols. Each of them can be true/false.

cs: robo at coffee shop
off: robo at Shyam's office
mr: robo at mail room
lab: robo at AI lab

Following are the various *states*' variables of the world, which can also be true/false:

mw: Mail waiting in the mail room
rhc: Robo is holding coffee
swc: Shyam wants coffee
rhm: Robo is holding mail

Absence of a state is represented by its negation. For example, $\overline{rhc}$ indicates that Robo is not holding coffee. Various *actions* performed by the Robo are

mc: Robo moves clockwise
mcc: Robo moves counterclockwise
puc: Pick up coffee
dc: Deliver coffee
pum: Pick up Mail
dm: Deliver Mail

Presence of an action symbol indicates that the action is true.

A state (of the world) may comprise many parameters or preconditions for the action to take place at that state. For example, the state,

$$\langle lab, \overline{rhc}, swc, \overline{mw}, rhm \rangle \tag{15.1}$$

indicates that the Robo is in an AI lab, Robo has no coffee in hand, Shyam wants coffee, mail is not waiting, and Robo holds the mail. Another state,

$$\langle lab, rhc, swc, \overline{mw}, \overline{rhm} \rangle, \tag{15.2}$$

indicates that the Robo is in the lab, Robo is holding coffee, Shyam is waiting for coffee, mail is waiting in the mail room, and Robo is not holding the mail. Table 15.3 shows the transitions for certain states. For example, in the third row, we note that after performing the action *dm* (deliver mail), in new state created, the state $\overline{rhm}$ indicates that variable "Robo is holding mail" is false.

Table 15.3 Some mapping: State × Action → State, for Fig. 15.2

State	Action	Resulting state
$\langle lab, \overline{rhc}, swc, \overline{mw}, rhm \rangle$	*mc*	$\langle mr, \overline{rhc}, swc, \overline{mw}, rhm \rangle$
$\langle lab, \overline{rhc}, swc, \overline{mw}, rhm \rangle$	*mcc*	$\langle off, \overline{rhc}, swc, \overline{mw}, rhm \rangle$
$\langle off, \overline{rhc}, swc, \overline{mw}, rhm \rangle$	*dm*	$\langle off, \overline{rhc}, swc, \overline{mw}, \overline{rhm} \rangle$
…	…	…

15.4 Forward Planning

One of the simplest plannings is to treat the planning as a path planning problem in the state-space graph. The nodes here are states, transitions are actions, and results of actions are also states. A forward planner searches the state-space graph from the start state for the goal state. Figure 15.3 shows the state-space graph for forward planning with the start state as $\langle cs, \overline{rhc}, swc, mw, \overline{rhm}\rangle$, with three transitions from start state, corresponding to the actions: pick up coffee (*puc*), Robo moves clockwise (*mc*), and Robo moves counterclockwise (*mcc*). If we choose the action *puc*, the next state is $\langle cs, rhc, swc, mw, \overline{rhm}\rangle$. The new state indicates that the Robo still remains facing the coffee shop; since it picked up coffee, the robot holding coffee is no more false, mail waiting remains true (unchanged). The new states as a consequence of the various actions are self-explanatory, and are similar to this description [2, 13].

Note that, being closed reasoning, in the world of actions we explicitly specify each of the variable as True or False.

The branching factor in Fig. 15.3 is 3, and the search can be done in DFS or BFS. Theoretically, since, the Robo can be at any of the four locations, and the other four parameters in a state can be true/false, there are $4 \times 2 \times 2 \times 2 \times 2 = 64$ total possible states in the world. Obviously, all of these states are not possible to reach in a graph search.

The representation above is simple and clear, but it is not suitable due to following reasons:

- there are too many states to acquire, reason, and represent,
- small change in the requirements will need a major change in the model, for example, if we need to have information about robot battery level to be added as one of the parameters, the entire structure gets modified.

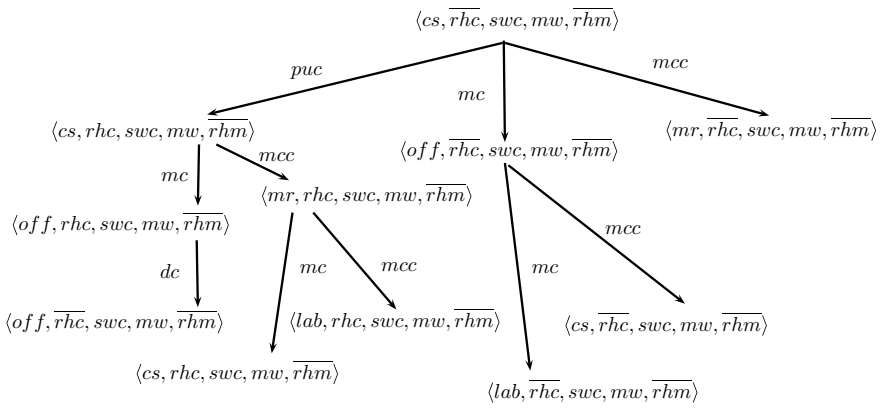

Fig. 15.3 State space for forward planning

The improvement in complexity is possible, and can be based on the following criteria: we note that the actions have a structure and that can be used to make actions compact. We also note that a precondition of an action should be true before the action takes place. For example, the action of the Robo to pick up the coffee (puc) requires the precondition of "Robo's location is coffee shop and Robo does not hold the coffee", which is expressed as $cs \wedge \overline{rhc}$. This means that *puc* is not available at other preconditions (or constraints) [11].

15.5 Partial-Order Planning

A *partial-order planning* is a search carried out by refining the partial plans through addition of actions and orderings. Alternatively, a search may proceed through performing abstract actions in the plan, by replacing fragments capable of carrying out those actions—the planning is called *hierarchical planning*. It has been now well understood that even the state-space search methods are nothing but other ways of refining partial plans. This refining is done by growing the prefix or suffix of the plan and different ways of refining a partial plan can be suitably interleaved.

The forward planner enforces a *total ordering* on actions at all the stages in the planning process. The idea of partial ordering between actions only commits to an ordering between actions when forced. A partial ordering is a "$\leq$" relation, that is, *reflexive*, *transitive*, and *antisymmetric*. It is the set of actions together with partial ordering, representing a "before relation" on actions, such that any total ordering of the actions, consistent with partial ordering, will solve the goal from the initial state. That is, $Act_0 < act_1$ means action Act_0 appears before the Act_1 in the partial order.

A partial-order plan comprises the following sets:

1. *Set of steps.* Each step maps to some operator, except the start and end step. There are no preconditions for start step and start state is its post-condition. Similarly, the final step has the goals as its preconditions and has no post-conditions.
2. *Set of orderings of steps.* Each ordering is a pair of steps, in the form of previous and next step. The start step is ordered first of all, and the finish step is ordered after all the steps.
3. *Set of causal links.* Each causal link is a pair of steps and a proposition, which is a post-condition of the first step and a precondition of the second. The first step is always ordered before the second step.

In fact, a partial planner works as follows: begin with actions *start* and *finish*, and with partial order $start < finish$. The planner maintains an agenda set of $\langle P, A\rangle$ pairs, where, A is action in plan, and P is the *precondition* of A. First, $\langle G, finish\rangle$ is chosen, such that G is the precondition for $Goal$. Then at each stage, a pair $\langle P, act_1\rangle$ is chosen from the agenda, where P is the precondition for action act_1. Subsequent to this, act_0 is chosen to achieve P, which is either already in the plan or it is *start* to achieve P, or it could be a new, that is, added to the plan. The act_0 must occur before

act_1, that adds a new causal link. Any action to delete P must happen after act_1 or before act_0. If act_0 is a new action, its preconditions are added to the agenda, and the process continues till the agenda is empty.

15.6 Planning Languages

The STRIPS formulation for planning problems uses a simple and general format for specifying operators with clear semantics. It employs propositional logic as a language for describing states of the world, with a set of conditions and Boolean variables to describe states. A complete assignment maps the set of conditions to possible values. For example, truth values and a partial assignment maps a subset of conditions to values, and a complete assignment is a state. An operator comprises two *partial assignments*—first is *preconditions* or *results*, which decide the states for applying the operator. The second is *post-conditions*, that decide the next state resulting when an operator is applied in a particular start state. In addition, the implicit frame axioms indicate that the value of any condition, which is not mentioned in an operator's post-conditions, is unchanged due to the application of an operator.

A planning problem instance consists of a set of operators, an Initial state, and a Goal. The Goal may be a Boolean formula also. Generally, a solution is in the form of a partially ordered multi-set of operators, which satisfies the following condition: any total ordering consistent with the given partial order transforms the initial state assignment into a new assignment that satisfies the Goal formula. The STRIPS formulation provides the semantic foundations for many extensions, including those handling external events, multiple agents, probabilistic transformations, and variants that allow the agent to observe aspects of the current state and choose actions conditioned on observations.

The STRIPS is action-centric representation, which is based on the idea that most things are not affected by single action. It specifies *action*, *precondition*, and *effect*. For example, for the goal: "Robo to pick up coffee", we write

$$\begin{aligned} &\mathit{precondition} : \mathit{cs} \wedge \overline{\mathit{rhc}} \\ &\mathit{effect} : \mathit{rhc}. \end{aligned}$$

The other features are unaffected in the above. The action of delivering coffee (dc) is

$$\begin{aligned} &\mathit{precondition} : \mathit{off} \wedge \mathit{rhc} \\ &\mathit{effect} : \overline{\mathit{rhc}} \wedge \overline{\mathit{swc}}. \end{aligned}$$

Apart from the above, you need to specify the initial states and goal.

The *action* or the *rule* has two formats:

– *Causal Rule*, when a feature gets a new value, and
– *Frame Rule*, when a feature keeps its value.

We can demonstrate the change of Robo's location (see Fig. 15.2), from current $RLoc^0$ to a new location $RLoc^1$ on account of action *mcc*, *mc*, and one which is neither *mcc* nor *mc*, as follows:

$$\begin{aligned}
&(RLoc^1 = cs) \leftarrow (RLoc^0 = off) \wedge (Act = mcc)\\
&(RLoc^1 = cs) \leftarrow (RLoc^0 = mr) \wedge (Act = mc)\\
&(RLoc^1 = cs) \leftarrow (RLoc^0 = cs) \wedge (Act \neq mcc) \wedge (Act \neq mc).
\end{aligned} \tag{15.3}$$

In the above formulas, the first two are causal rules, and the last one is the frame rule. Similarly, the state "Robo holds coffee" in the resulting state would depend on whether it was holding coffee in the previous state and its action:

$$\begin{aligned}
&rhc^1 \leftarrow rhc^0 \wedge Act \neq dc\\
&rhc^1 \leftarrow Act^0 = puc
\end{aligned} \tag{15.4}$$

where the first is frame rule, and the second is the causal rule.

15.6.1 A General Planning Language

STRIPS is a problem-solving program implemented in LISP and has been used in the application of robotic research; it is a member of the class of problem solvers that search a space of *world models* to find a model through which the goal is achieved. We assume that, for some world model, there exists a set of operators; each of such operators transforms the world model into some other world model. Having this, the task of the problem solver is to determine a composition of operators that transform a given initial world model into one that satisfies the goal condition [3].

The primary objectives in robotics-based class of problems are rearranging physical objects, and navigation of robot—problems that require general world models, and are more complex than those used in the solution of puzzles and games. Usually, a list or simple matrix structures is adequate to represent a state of such problems. The world model for a robot problem solver must comprise a large number of facts and relations about the position of the robot, and about the positions and attributes of objects, open spaces, and boundaries. In STRIPS, the world model is represented by a set of well-formed formulas in the first-order predicate logic (FOPL) [4].

A solution is built using operators, which are the basic elements of general robotics planning language. For a robot problem, each operator corresponds to an action routine whose execution causes a robot to take that action. For example, we might have a routine to push back if the robot touches the wall, or a routine to lift and object, or to grip an object, and so on, there are a large number of routines that correspond to actions of a robot.

15.6.2 The Operation of STRIPS

The problem space for STRIPS language comprises the initial world model, a set of available operators with their effects on the world model, and a goal statement. The world model is represented by a set of well-formed formulas (wffs) of FOPL, e.g., to describe a world model in which the robot is at location a and boxes B, C are at locations b, c respectively, we would use the following wffs to express this knowledge [4]:

$$ATR(a)$$
$$AT(B, b)$$
$$AT(C, c).$$

We might also use the wffs to express the general rule that an object u at place x is not in a place y, i.e., an object cannot exists at two places.

$$(\forall u\ \forall x\ \forall y)\{[AT(u, x) \wedge (x \neq y)] \Rightarrow \neg AT(u, y)\}. \tag{15.5}$$

We can represent a complex world model using the first-order predicate logic, and can use standard theorem-proving programs to answer questions about the model. The operators are grouped into families of operators, called *schema*. For example, an operator *goto* for moving the robot from one point m on the floor to other point n is a schema. For this, distinct operators (one for each pair of points) are grouped into a family of *goto* operators, and $goto(m, n)$ represents a move from the initial position m to the final position n. The members of *goto* schema are $goto(m, a_1)$, $goto(a_1, a_2)$, ..., $goto(a_k, n)$. In STRIPS, specific constants will already have been chosen for the operator parameters when an operator is applied to a world model.

First of all, it is necessary to determine whether or not there is an instance of an operator schema applicable to the current world model. It is required that an instance of the corresponding wffs (well-formed formulas) schema exists, and it logically follows from the model. Each operator schema is defined by a description having two parts: conditions under which the operator can be applied, and the effects of application of that operator. The precondition for an operator schema is represented as a wff. The effects of application of an operator are defined by a list of wffs that must be added to the model, and a list of wffs that are no longer true, hence must be deleted. As an example, consider the question of applying instances of the operator subschema $goto(m, b)$ to a world model containing the wff $ATR(a)$. If the precondition wff schema of $goto(m, n)$ is $ATR(m)$, then we note that the instance $ATR(a)$ can be proved from the world model. Thus, an applicable instance of $goto(m, b)$ is $goto(a, b)$.

It is important to understand the difference between the parameters in wff schema, and existentially and universally quantified variables. These variables are used in FOPL formula in theorem-proving programs (e.g., Resolution theorem) that would handle wff schema. For example, in resolution, $\forall x \forall y\ goto(x, y)$ will have substitution

$\{a/x, b/y\}$, to make it a clause, whereas in STRIPS there is a chain of pairs of (x, y) to implement $goto(x, y)$, hence it will require some modifications.

A goal statement is also in the form of a wff, e.g., a task of moving the boxes B and C to locations d, e, respectively, might be expressed a goal as

$$AT(B, d) \wedge AT(C, e).$$

In summary, the problem in state space for STRIPS comprises three entities:

1. *Initial world model.* It is a set of wffs to describe the present state of the world.
2. *Set of operators.* The operators and their description in the form of effects and preconditions as wffs.
3. *Goal.* It is a condition in the form of a wff.

The problem is taken as solved when a world model that satisfies the goal wff is deduced using the initial world model with the application of operators.

15.6.3 Search Strategy

A very simple problem-solving strategy is to apply all the applicable operators to the initial world model and create a set of successor models. Then, continue to apply all applicable operators to these successors and to their descendants generated through DFS or BFS search, until a model is deduced having the goal formula as a theorem. However, it is quite likely that in most real-word problems modeled using this approach, the number of operators applicable to any given world model will be too large. Hence, such a simple system would generate trees that are large in number as well as size. Hence, such world models would be impractical.

An alternative strategy is to extract differences between current world model and the goal model, with the objective as to how to move from one to another with minimum changes. Then find out the operators that may reduce the differences between these models. Once the relevant operators are found out, we try to solve the subproblem that produces a world model to which it is applicable. If such a model is found, then we apply the relevant operators and consider the original goal in the resulting model.

15.7 Planning with Propositional Logic

Consider the graph/tree shown in Fig. 15.4, with s_0 as start state, and $a_0 \ldots a_{n-1}$ as actions representing a path leading to the goal state s_n. If s_0, $a_0 \ldots a_{n-1}$, s_n are considered as propositional expressions, then Eq. [10],

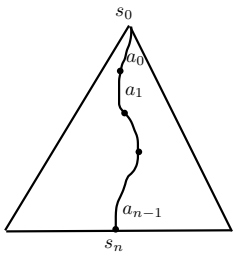

Fig. 15.4 Propositional planning

$$p = s_0 \wedge (a_0 \wedge a_1 \wedge \cdots \wedge a_{n-1}) \wedge s_n \tag{15.6}$$

is a propositional expression representing the path to goal state from start state. If there is only one goal, then p is the only path to goal, and all other propositions p' are false. Equation 15.6 can always be transformed into a *CNF* (conjunctive normal form) or *SAT* expression. Thus, planning through propositional logic is to find a satisfiability expression, comprising *start state*, *path*, and *goal state*, i.e., logical sentence equal to

$$\textit{initial state} \wedge \textit{all proposition action descriptions} \wedge \textit{goal}. \tag{15.7}$$

In other words, a model that satisfies a sentence will assign true to all actions that are part of a correct plan. If the planning problem is unsolvable then there is no sentence that is satisfiable. The following example demonstrates planning with propositional logic [12].

Example 15.3 Flight Planning.

Initial Plan: Let us assume that initially, the plane p_1 is at *DEL* (Delhi) and p_2 at *CAL* (Calcutta). The actions should correspond to flying these planes so that the goal: "p_1 at *CAL*, and p_2 at *DEL*" is satisfied.

The initial state (0) is represented by

$$At(p_1, DEL)^0 \wedge At(p_2, CAL)^0.$$

Since the propositional logic has no *closed world*, it is also necessary to show that initially the planes p_1, p_2 are not *CAL, DEL*, respectively, i.e.,

$$\neg At(p_1, CAL)^0 \wedge \neg At(p_2, DEL)^0.$$

The initial conditions correspond to start state s_0.

The goal also needs to be specified with a particular time step. Since it is not known how many time steps it would consume to iterate the action, a worst case time limit needs to be specified so that either the goal is reached within that time limit, else the solution is terminated as failure.

We test the following assertion for a goal at time $T = 0$ (at start),

$$At(p_1, CAL)^0 \wedge At(p_2, DEL)^0.$$

If that fails, apply certain actions and again try it at time $T = 1$, and so on, up to time $T = T_{max}$. This is shown in Algorithm 15.1, where for every iteration, for the next value of T (time), the problem is translated to *SAT* (satisfiability) problem using the procedure *translate-to-SAT*. This results in *CNF* expression and a mapping of the solution to the problem. The *SAT-Solver* procedure returns the assignment for the above *CNF* expression. If the assignment is satisfying the solution (i.e., not null), the solution is extracted for the present mapping, and assignment is returned as result. If this does not happen, the procedure is iterated as per the T_{max} iteration time [8].

Algorithm 15.1 Planning with Propositional Logic

```
INPUT : AplanningProblem;
                T_max: an upper limit for plan length
for T = 0 to T_max do
   cnf , mapping ← translate-to-SAT(problem, T)
   assignment ← SAT-Solver(cnf)
   if Assignment is not null then
      Return Extract-solution(asignment, mapping)
   end if
end for
Return Failure
End
```

15.7.1 Encoding Action Descriptions

We have a propositional symbol for each occurrence. For the plane p_1 to be at *CAL*, there is a proposition:

$$At(p_1, CAL) \Leftrightarrow (At(p_1, CAL)^0 \wedge \neg fly(p_1, CAL, DEL)) \vee (At(p_1, DEL)^0 \wedge fly(p_1, DEL, CAL)^0) \quad (15.8)$$

There ought to be a plan that tries to achieve the goal at $T = 1$. Now suppose CNF is

$$Initial\ state \wedge successor\ state\ axioms \wedge goal^1,$$

that is, goal is true at $T = 1$, we check and verify that

$$fly(p_1, DEL, CAL)^0 \wedge fly(p_2, CAL, DEL)^0$$

is the model, and other assignments are false.

15.7.2 Analysis

For the proposition *fly* having general form as $fly(p, o_1, o_2)$, with time T-steps, the number of propositions represented by $|p|$, and objects (o_1, o_2, etc.) as O, the complexity expression is given by,

$$\begin{aligned} fly(p, o_1, o_2) &\Rightarrow T \times |p| \times |o_1| \times |o_2| \\ &\Rightarrow T \times |p| \times |O|^2 \\ &\Rightarrow T \times |Act| \times |O|^p \end{aligned}$$

where T is the number of time steps, p is *arity* of function (here it is 2), O is the number of objects o_1, o_2, etc. We note that the complexity is exponential. The term $|Act|$ is for the action, like *fly*, i.e., how many predicates are there in total [8].

15.8 Planning Graphs

The planning graphs give better heuristics and consist of a sequence of levels, for *time steps* in plan. Each level has literals (constant values) which have become true because of the previous action, and each level has preconditions for the next action. The planning graphs represent the *actions* as well as *inactions*. The following example demonstrates the application of planning graphs [5, 12].

Example 15.4 Problem of the solution to "have a Pizza and eat Pizza".

$$\begin{aligned} &init(have(pizza)) \\ &Goal(have(pizza) \wedge eaten(pizza)) \\ &Action(eat(pizza) \\ &\quad Precond : have(pizza) \\ &\quad Effect : \neg have(pizza) \wedge eaten(pizza)) \\ &Action(cook(pizza) \\ &\quad Precond : \neg have(pizza) \\ &\quad Effect : have(pizza)). \end{aligned}$$

The planning graph for these actions is shown in Fig. 15.5. The box action in the figure indicates the *mutual exclusions* of actions.

In the planning graph, all the actions A_i at level i contain all actions that are applicable at state S_i, along with constraints saying which part of the actions cannot be executed. Every state at level S_i contains all literals that could result from any choice of actions at A_{i-1}, along with constraint saying which part of actions cannot be executed.

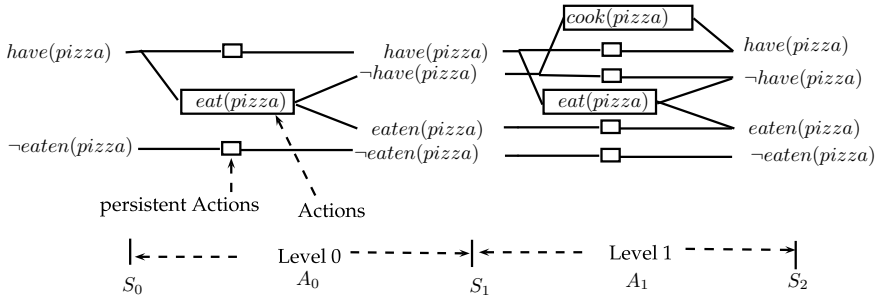

Fig. 15.5 A planning graph for "To have pizza and eat pizza"

We note that graph plan provides lesser complexity because it does not require choosing about all actions. It just records the impossibility of certain choices using *mutex*, i.e., through mutual exclusion links. For example, when $\neg have(pizza)$ is chosen, the action $have(pizza)$ is excluded. This results in the complexity as a function of low polynomial actions and literals [11].

15.9 Hierarchical Task Network Planning

In the hierarchical task planning, each level of hierarchy is decomposed into smaller levels. It is common for areas like military mission, administration, program development, where a task is reduced to small number of activities at the next level, so that the computational effort of arranging those activities is low. This results in reduction in complexity to linear time from the original exponential [13].

Consider an example of "building a house", where the task of house building can be decomposed to acquiring land, preparation of design map, obtaining the NOC (no objection certificate) from municipal corporation, arranging for a house loan, hiring a builder, paying the builder, and so on, as shown in Fig. 15.6.

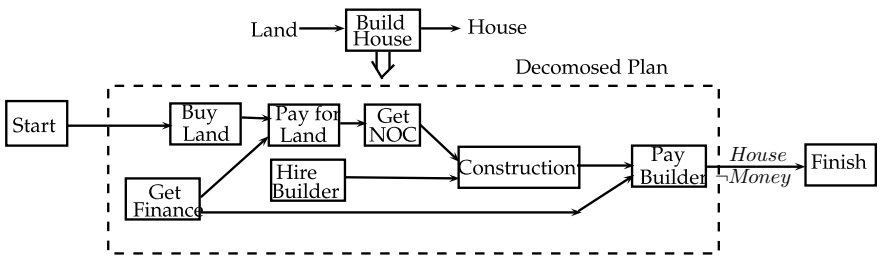

Fig. 15.6 Decomposition of task of house building

We understand that some of these activities can be done in parallel, but not all. Thus, there is a partial-order relation between those actions. The decomposition can be expressed as $decompose(a, d)$, where action a is decomposed into a partial-order plan d. Various activities for building a house as per the plan in Fig. 15.6 can be formally described as follows.

$$Action(Buildhouse,\ Precond : BuyLand,\ Effect : House) \tag{15.9}$$

$$Action(Buyland,\ Precond : Money,\ Effect : Land \wedge \neg Money) \tag{15.10}$$

$$Action(Getfiance,\ Precond : Goodcredit,\ Effect : Money \wedge Mortgage) \tag{15.11}$$

$$Action(Hirebuilder,\ Precondition : Nil,\ Effect : contract) \tag{15.12}$$

$$Action(Construction,\ Precond : NOC \wedge Builderhired,\ Effect : Housebuilt \wedge \neg NOC) \tag{15.13}$$

$$Action(Paybuilder,\ Precond : Money \wedge Housebuilt,\ Effect : \neg Money \wedge house \wedge \neg Contract) \tag{15.14}$$

$$Decompose(steps : \{A_1 : Get\ NOC,\ A_2 : Hirebuilder,\ A_3 : construction, A_4 : Paybuilder\} \tag{15.15}$$

$$Orderings : \{start \prec A_1 \prec A_3 \prec A_4 \prec Finish;\ Start \prec A_2 \prec A_3\}. \tag{15.16}$$

Here $A_i \prec A_j$ indicates that activity A_i precedes the activity A_j.

The linking of states and activities through activities/resources for the house building plan can be expressed as follows:

$$Links : \{start \xrightarrow{Land} A_1, Start \xrightarrow{Money} A_4, A_1 \xrightarrow{NOC} A_3,\ A_2 \xrightarrow{Contract} A_3,\ A_3 \xrightarrow{Housebuilt} A_4, A_4 \xrightarrow{House} Finish,\ A_4 \xrightarrow{\neg Money} Finish.\} \tag{15.17}$$

15.10 Multiagent Planning Systems

There are a number of good reasons for having multiple agents creating plans:

1. The agents may represent real-life entities, which may require the privacy and autonomy also to be maintained.
2. Changing of an existing distributed system to a multiagent system is less costly than to a centralized system .
3. Creating and maintaining plans locally allows for more efficient reaction, especially when communication is limited, and
4. Dividing a planning problem into smaller pieces and solving those in parallel turns out to be many times more efficient. This is true particularly when the individual planning problems are not tightly coupled.

In spite of many benefits of multiagent systems, following are the challenges in developing multiagents planning:

1. How to put additional constraints upon the agents before planning, such that their resulting plans can still be coordinated?
2. How to efficiently construct plans in a distributed environment?
3. How to make collaborative decisions when there are multiple options, and agents have their own preferences for these options?
4. In what condition should a planning agent ask for more specific information to the user?
5. How to find out the magnitude of privacy lost in the process of coordinating plans?

Definition 15.2 Multiagent planning problem.

In general, a multiagent planning problem is a problem of planning by and for a group of agents. Except for centralized multiagent planning problems, each agent in such a problem has in fact a private, individual planning problem. A typical individual planning problem of an agent includes a set of operations with some costs attached, and pre- and post-conditions that it can perform, and a set of goals (with reward values), and the current (initial) state of this agent. The following statement captures the concept of multiagent planning:

$$Multiagent\ planning = planning + coordination. \tag{15.18}$$

□

The solution to a multiagent planning problem is a plan: a partially ordered sequence of actions that, when executed successfully, results in a set of achieved goals for some of the agents. Most techniques can deal with problems where the actions and goals of the agents are 1. only weakly dependent upon each other, 2. agents are cooperative, and 3. communication is reliable. However, in general a multiagent planning problem may encounter a lot variety of situations along these three axes. Some characteristics are described below.

1. From independent to strongly related.
   a. *Independent:* There are no shared resources, and no dependencies. For example, a robot is lifting a box.
   b. *Strongly related:* They have joint actions, and the resources are shared. For example, car assembly is a case of strongly related.
2. From *cooperative* to *self-interest* agents: The participating agents interested in optimizing their own utility is the case of self-interesting agents, while the agents involved in supply chain management is an example of cooperative.
3. From *no communication* to *reliable communication*: In hostile environments agents may not or cannot communicate during execution. This may require agents to be equipped in advance, with or without some initial communication before the execution starts. The robots rescuing people in disaster scenarios, or working on a inter-planetary exploration mission, are examples of the first type, while agents working in a supply chain management is an example of the second category.

15.11 Multiagent Planning Techniques

Multiagent planning techniques cover quite a range of solutions to different phases of the problem. In general, the following *phases* can be distinguished in task sharing:

1. Allocate goals to agents.
2. Refine goals into subtasks.
3. Schedule subtasks by adding resource allocation including the agents, and timing constraints.
4. Communicate planning choices (i.e., prior steps) to recognize and resolve conflicts.
5. Execute the plans.

Planning is a combination of phases 2 and 3 in the above, which are often interleaved. Any of these steps could be performed by one agent or some subset. Not all the above phases of general multiagent planning process need to be considered in every multiagent planning problem. For example, there is no need for phase 1 if there are no common or global goals for the multiagents. Some multiagents system may combine different phases, for example, while constructing their plans agents may coordinate with each other due to the combination of phases 2, 3, and 4. Alternatively, robots may postpone coordination until the execution phase (i.e., combining phase 4, 5). This may result, for example, when they unexpectedly encounter each other while following their planned routes.

It is possible in general, to interleave any combination of five phases listed above, depending on the nature of the planning problem, resulting in a wide variety of possible problem sub-classes. In the following we present the phases in more detail.

15.11.1 Goal and Task Allocation

Centralized methods often take care of the assignment of goals and tasks to agents during planning. There are, however, many other methods to assign tasks in a more distributed way, giving the agents a higher degree of autonomy and privacy. For example, complex task allocation protocols may be used, or auctions and market simulations can also be used.

One way to assign the tasks to agents is through the method of auction, which is a way to assign a task to that agent who attaches (or bids) the highest value or the lowest cost of performing it, called *private value*. A protocol called *Vickrey* auction is a frequent example in multiagent systems, where each agent makes a closed bid, and the task is assigned to the highest bidder, but not on the price of that bidder, but for the price of the second highest bidder! The Vickery protocol has the good property due to which the bidding agents will simply bid their true private values, with no need for additional reasoning about it is worth for the others.

Economics and market simulations can also be the basis for allocation of resources among agents. For example, it is shown how costs can be turned into a coordination device. These methods are useful for task assignment (phase 2), and also for the coordination of agents after plan construction (phase 5).

In concerning the value-oriented environments, these game-theoretical approaches become more important where agents reason about cost of their decision-making (or communication).

15.11.2 Goal and Task Refinement

Task assignment can be done through a single agent, using Hierarchical Task Networks or nonlinear planning. More than one planner with more sophisticated models of temporal extent can be introduced, along with centralizing as well as combining the phases 2 through 4.

15.11.3 Decentralized Planning

Instead of one agent planning for rest of the agents, the second and third phases can be implemented through local planning by each of the agents. In principle, any planning technique can be used in this condition, and different agents may even use different techniques. Some of the approaches join the individual plannings (phases 2 and 3), along with the coordination of the plans (phase 4).

A distributed version of a planner can be used to integrate phases 1 through 4, to plan for a single agent in parallel.

In one setting, each agent has a partial knowledge of the plans of other agents using some specialized plan representation techniques. In such kind of environment, coordination is achieved as follows: If an agent A informs the other agent B about part of its own plan, then agent B merges this information into its own partial global plan. The agent B can then try to improve the global plan by, for example, eliminating redundancy in the plan, and this improved plan is shown to the other agents, who might reject, or accept, or modify it. This process may run concurrently with the execution of the (first part of the) local plan.

15.11.4 Coordination After Planning

One of the tasks is the planning of coordination after plans are constructed on individual basis (phase 4), called *plan merging* . Its objective is construction of a joint plan for a set of agents, given the individual plans of each of the participating agents. In coordination planning, every pair of agents helps each other by changing the state of the world such that the conditions of the other agent become satisfied. In this process, changing the state of the world may be helpful to these two, but may also interfere with the correct conditions of the remaining $n - 2$ agents, assuming a system of n agents.

To specify the constraints on plans, one approach is to use propositional temporal logic, which will ensure that only feasible states of the environment are reached. A theorem prover algorithm generates a sequences of communication actions, on receiving these constraints. In fact, these communication actions implement semaphores that guarantee that no event will fail. For resolving conflicts, restrictions are required to be introduced on individual plans in phase 3, which will also ensure efficient merging.

A different approach to plan merging uses the distributed approach to improve social welfare, based on the sum of the benefits of all agents. This approach uses a process of group constraint aggregation, where agents construct an improved global plan by voting for joint actions, in an incremental way. The agents even propose algorithms to deal with insincere agents, and to interleave planning, coordination, and execution [11].

15.12 Summary

When an agent is interested in controlling the evolution of its environment, there is a need for planning. Thinking as an algorithm, a planning problem has an input in the form of possible courses of actions, a predictive model for the required dynamics, and a measure for performance to evaluate the courses of actions. The output or solution of this algorithm is one or more courses of actions that satisfy the specified requirements for performance.

The agents are classified as *reactive* and *planning agents*.

The classical planning problem is to plan reaching the goal state(s) from the initial state, for a given set of actions. A majority of research in planning is toward planning in the environments that are *dynamic*, *stochastic*, and *partially observable*. To carry out this, the existing classical planning techniques are extended to allow interleaving of planning and scheduling.

Automated planning techniques are being applied in many domains, that include robotics, process planning, web-based information gathering, autonomous agents, and spacecraft mission controls. In automated planning, a solution to a problem can be described in terms of a sequence of steps that transforms some initial description of the problem state, for example, the initial configuration of a puzzle, into a description satisfying a specified goal criterion. For a simple automated planning problem, called *classical planning problem*, the following assumptions are made:

1. the actions are deterministic,
2. the world is fully observable, and
3. the closed world assumption.

Agents are classified according to the techniques they employ in decision-making:

1. Reactive agents, and
2. Planning agents.

Forward planning, one of the simplest planning, is path planning where nodes are states, transitions are actions, and results of actions are also states. A forward planner searches the state-space graph from start state for goal state. One problem with forward planner is its time complexity, which is exponential.

A *partial-order plan* consists of (1) a set of steps, each mapping to an operator, (2) a set of orderings, and (3) a set of causal links.

STRIPS is an action-centric language, based on the idea that most things are not affected by a single action. It specifies *action*, *precondition*, and *effect*. The problem space for STRIPS is defined by an initial world model, a set of operators, and a goal condition state.

Planning through *propositional logic* is to find a satisfiability expression, which can be given as

$$\textit{initial state} \wedge \textit{all proposition action descriptions} \wedge \textit{goal}.$$

Other approach for automated planning is *planning graphs*, which consists of a sequence of levels, for *time steps* in plan, representing *actions* as well as *inactions*.

In the areas such as military mission, administration, program development, where a task is reduced to a small number of activities at the next level, such that the computational effort of arranging those activities is low, the *hierarchical task planning* using networks is preferred.

Splitting a planning problem into smaller problems and solving these in parallel turns out to be more efficient, thus motivating the use of multiple agents for creating plans. Multiagent planning covers the following *phases*:

1. Allocate goals.
2. Refine goals into subtasks.
3. Add resource including the agents.
4. Communicate planning choices.
5. Execute the plans.

Exercises

1. Consider the standard Towers of Hanoi problem with 3 pegs and 4 number of disks (d_1, d_2, d_3, d_4, with d_1 at the top). The disks are to be transferred from start-peg to end-peg, using intermediate peg, one at a time such that at no time larger disk comes over the smaller. The disk d_1 is the smallest and d_4 is the largest. Make use of only 3-predicates: unary predicate: *clear*, and binary predicates: *on* and *smaller*, and only one action: *puton*(x, y) needs to be used.
   Write the domain of the problem, and make use of *forward planning* to plan the solution to move all the 4 disks from start-peg to end-peg.
2. Given the 3-SAT problem:

$$(\neg p_1 \vee p_2 \vee p_3) \wedge (p_1 \vee \neg p_2 \vee p_3) \wedge (p_1 \vee p_2 \vee \neg p_3),$$

   solve it using *forward planning*. (Hint: you need to assume some *operators* (i.e., actions) to assign the values to variables $p_1 \ldots p_3$.)
3. Given the Table T, and blocks A, B, C, D, having different positions on the table, apply STRIPS to plan the solution of the following problem:
   Initial state **I** as

$$clear(A), clear(B), clear(C), clear(D),$$
$$on(A, T), on(B, T), on(C, T), on(D, T),$$

   i.e., the blocks $A \ldots D$ are on the table, and their tops are clear.
   Final State **F** :
   $on(A, B), on(B, C), on(C, D), on(D, T), clear(A)$.
   Use the action *puton*(X, Y), $x \neq y$, where X is a block $A \ldots D$ and Y is either table T or block $A \ldots D$. Give the forward planning to reach state **G** starting with state **I**.
4. Use STRIPS for planning of the following problem: You are at home, and you have money, and you are required to buy milk. Assume the necessary start and goal states, actions, preconditions, and results for this planning job.
5. Give the STRIPS representations to actions: pick up mail and deliver mail (ref. Fig. 15.2).
6. Suppose the robot (in Fig. 15.2) cannot carry both coffee and mail at the same time. Make use of some constraints to provide the planning for this situation.

Assume that the robot can carry a box in which it can place objects, so that it can carry the box and the box can hold the mail and coffee.

7. Modify the problem in Fig. 15.2, so that the robot has the work of cleaning the four rooms (mail room, office, coffee shop, lab). Assume that it will clean the room only when the room is unclean, and will not consume more than one rotation *mcc* or *mc* to reach any of these rooms.
8. Using the method of hierarchical task network planning, provide the automated planning for the following problems:

   a. Shopping grocery items from market.
   b. Deliver a lecture of AI.
   c. Robot path planning to cover the diagonal in a room.

9. Assume that you have three operators:

   f_1 : Precondition: a; effect: $\neg a \wedge b$
   f_2 : Precondition: $a \wedge c$; effect: $\neg a \wedge b \wedge \neg c$
   f_3 : Precondition: $b \wedge c$; effect: $\neg c \wedge d$

   Show the first three layers (proposition, action, and proposition) of the graph plan when the initial state is $a \wedge c$ (a and c both are true). Include the mutual exclusions and justify each of them.

References

1. Bborrajo D et al (2015) Progress in case-based planning. ACM Comput Surv 47(35):1–35
2. Bonet B, Geffner H (2001) Planning as heuristic search. Artifi Intell 129:5–33
3. Dean T (1996) Automated planning. ACM Comput Surv 28(1):85–88
4. Fikes RE, Nilsson NJ (1971) STRIPS: a new approach to the application of theorem proving to problem solving. Artifi Intell 2:189–208
5. Jonsson P et al (2000) Towards efficient universal planning: a randomized approach. Artifi Intell 117:1–29
6. Kaelbling LP et al (1998) Planning and acting in partially observable stochastic domains. Artifi Intell 101:99–134
7. Kambhampati S (1995) AI planning: a prospectus on theory and applications. ACM Comput Surv 27(3):334–336
8. Kautz H, Selman B (1992) Planning as satisfiability. In: Proceedings of the 10th European conference on artificial intelligence (ECAI 92), Vienna, Austria
9. Leckie C, Zukerman I (1998) Inductive learning of search control rules for planning. Artifi Intell 101:63–98
10. Melis E, Siekmann J (1999) Knowledge-based proof planning. Artifi Intell 115:65–105
11. Nareyek A (2005) Constraints and AI planning. Intell Syst 03(04):2005
12. Oh SC et al (2005) A comparative illustration of AI planning-based web services composition. ACM SIGecom Exchanges 5(5):1–10
13. Russell SJ, Norvig P (2005) Artificial intelligence—a modern approach, 2nd edn, Pearson