PLANNING

In which we see how an agent can take advantage of the structure of a problem to construct complex plans of action. The task of coming up with a sequence of actions that will achieve a goal is called planning.

We have seen two examples of planning agents so far: the search-based problem-solving agent and the logical planning agent

CLASSICAL PLANNING ENVIRONMENT

environments that are fully observable, deterministic, finite, static (change happens only when the agent acts), and discrete (in time, action, objects, and effects). These are called classical planning environments

Definition of Classical Planning

Classical Planning is the planning where an agent takes advantage of the problem structure to construct complex plans of an action. The agent performs three tasks in classical planning:

- Planning: The agent plans after knowing what is the problem.
- Acting: It decides what action it has to take.
- Learning: The actions taken by the agent make him learn new things

A language known as PDDL(Planning Domain Definition Language) which is used to represent all actions into one action schema.
PDLL describes the four basic things needed in a search problem:

- Initial state: It is the representation of each state as the conjunction of the ground and functionless atoms.
- Actions: It is defined by a set of action schemas which implicitly define the ACTION() and RESULT() functions.
- Result: It is obtained by the set of actions used by the agent.
- Goal: It is same as a precondition, which is a conjunction of literals (whose value is either positive or negative).

There are various examples which will make PDLL understandable:

- Air cargo transport
- The spare tire problem
- The blocks world and many more

Complexity of the classical planning
In classical planning, there occur following two decision problems:

1. Plan SAT: It is the question asking if there exists any plan that solves a planning problem.
2. Bounded Plan SAT: It is the question asking if there is a solution of length k or less than it.

There are following advantages of Classical planning:

- It has provided the facility to develop accurate domain-independent heuristics.
- The systems are easy to understand and work efficiently.

The following points highlight the two main planning methods used to solve AI problems. The methods are: 1. Planning with State-Space Search 2. Goal Stack Planning.

Method # 1. Planning with State-Space Search:

The most straight forward approach is to use state-space search. Because the descriptions of actions in a planning problem specify both preconditions and effects, it is possible to search in either direction: forward from the initial state or backward from the goal, as shown in Fig. 8.5. We can also use the explicit action and goal representations to derive effective heuristics automatically.

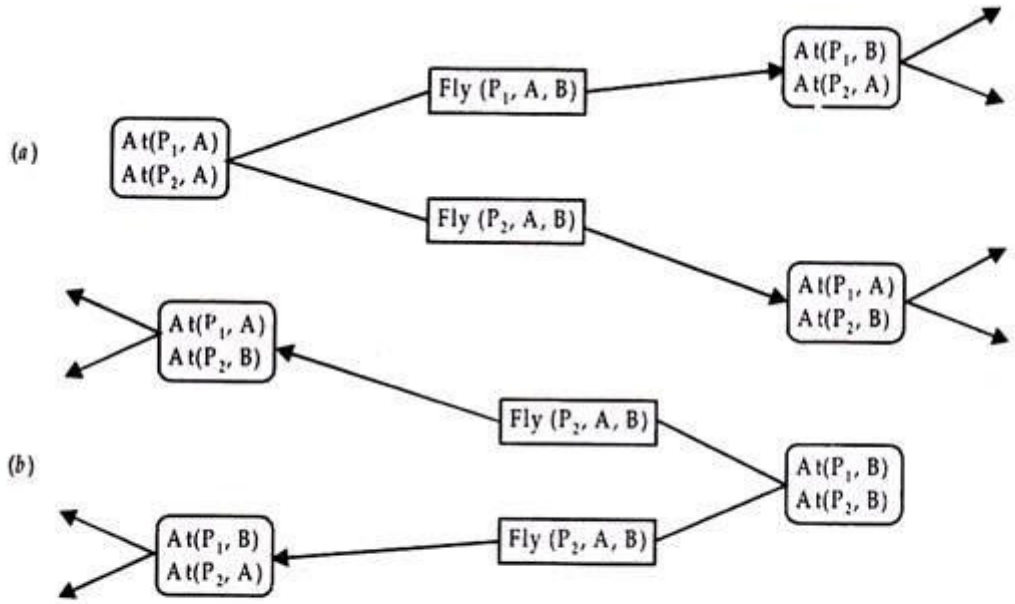


Fig. 8.5. *Two approaches to searching for a plan, (a) Forward (Progression) state-space search, starting in the initial state and using the problem's actions to search forward for the goal state, (b) Backward (regression) state-space search: a belief-state search starting at the goal state(s) and using the inverse of the actions to search backward for the initial state.*

10.1 DEFINITION OF CLASSICAL PLANNING

The problem-solving agent of Chapter 3 can find sequences of actions that result in a goal state. But it deals with atomic representations of states and thus needs good domain-specific heuristics to perform well. The hybrid propositional logical agent of Chapter 7 can find plans without domain-specific heuristics because it uses domain-independent heuristics based on the logical structure of the problem. But it relies on ground (variable-free) propositional inference, which means that it may be swamped when there are many actions and states. For example, in the Wumpus world, the simple action of moving a step forward had to be repeated for all four agent orientations, T time steps, and n2 current locations.

$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg At(p, from) \wedge At(p, to))$

$Init(At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK)$
$\wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2)$
$\wedge Airport(JFK) \wedge Airport(SFO))$
$Goal(At(C_1, JFK) \wedge At(C_2, SFO))$
$Action(Load(c, p, a),$
PRECOND: $At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg At(c, a) \wedge In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c, a) \wedge \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg At(p, from) \wedge At(p, to))$

Figure 10.1 A PDDL description of an air cargo transportation planning problem.

$Action(Fly(P_1, SFO, JFK),$
PRECOND: $At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$
EFFECT: $\neg At(P_1, SFO) \wedge At(P_1, JFK))$

10.1.1 **Example: Air cargo transport** Figure 10.1 shows an air cargo transport problem involving loading and unloading cargo and flying it from place to place. The problem can be defined with three actions: Load, Unload, and Fly. The actions affect two predicates: In(c, p) means that cargo c is inside plane p, and At(x, a) means that object x (either plane or cargo) is at airport a. Note that some care must be taken to make sure the At predicates are maintained properly. When a plane flies from one airport to another, all the cargo inside the plane goes with it. In first-order logic it would be easy to quantify over all objects that are inside the plane. But basic PDDL does not have a universal quantifier, so we need a different solution. The approach we use is to say that a piece of cargo ceases to be At anywhere when it is In a plane; the cargo only becomes At the new airport when it is unloaded. So At really means “available for use at a given location.”

The following plan is a solution to the problem:

[Load (C1, P1, SFO),Fly(P1, SFO, JFK), Unload(C1, P1, JFK), Load(C2, P2, JFK),Fly(P2, JFK , SFO), Unload(C2, P2, SFO)]

10.1.**2 Example: The spare tire problem**

Consider the problem of changing a flat tire (Figure 10.2). The goal is to have a good spare tire properly mounted onto the car's axle, where the initial state has a flat tire on the axle and a good spare tire in the trunk. To keep it simple, our version of the problem is an abstract one, with no sticky lug nuts or other complications. There are just four actions: removing the spare from the trunk, removing the flat tire from the axle, putting the spare on the axle, and leaving the car unattended overnight. We assume that the car is parked in a particularly bad neighborhood, so that the effect of leaving it overnight is that the tires disappear.

A solution to the problem is [Remove(Flat, Axle), Remove(Spare, Trunk),PutOn(Spare, Axle)]

$$
\begin{array}{l}
Init(Tire(Flat) \land Tire(Spare) \land At(Flat, Axle) \land At(Spare, Trunk)) \\
Goal(At(Spare, Axle)) \\
Action(Remove(obj, loc), \\
\quad \text{PRECOND: } At(obj, loc) \\
\quad \text{EFFECT: } \neg At(obj, loc) \land At(obj, Ground)) \\
Action(PutOn(t, Axle), \\
\quad \text{PRECOND: } Tire(t) \land At(t, Ground) \land \neg At(Flat, Axle) \\
\quad \text{EFFECT: } \neg At(t, Ground) \land At(t, Axle)) \\
Action(LeaveOvernight, \\
\quad \text{PRECOND:} \\
\quad \text{EFFECT: } \neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk) \\
\qquad \land \neg At(Flat, Ground) \land \neg At(Flat, Axle) \land \neg At(Flat, Trunk))
\end{array}
$$

Figure 10.2 The simple spare tire problem.

10.1.**3 Example: The blocks world BLOCKS WORLD**

One of the most famous planning domains is known as the blocks world. This domain consists of a set of cube-shaped blocks sitting on a table.2 The blocks can be stacked, but only one block can fit directly on top of another. A robot arm can pick up a block and move it to another position, either on the table or on top of another block. The arm can pick up only one block at a time, so it cannot pick up a block that has another one on it. The goal will always be to build one or more stacks of blocks, specified in terms of what blocks are on top

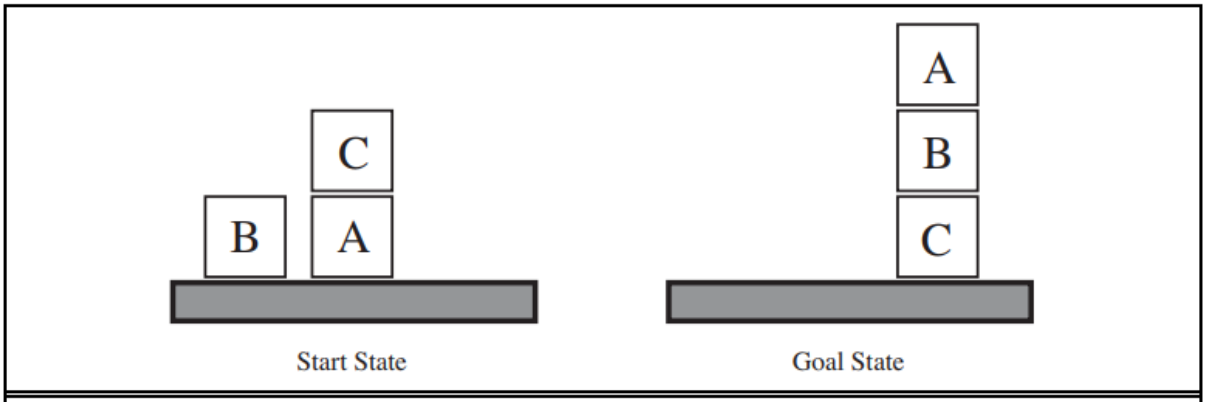
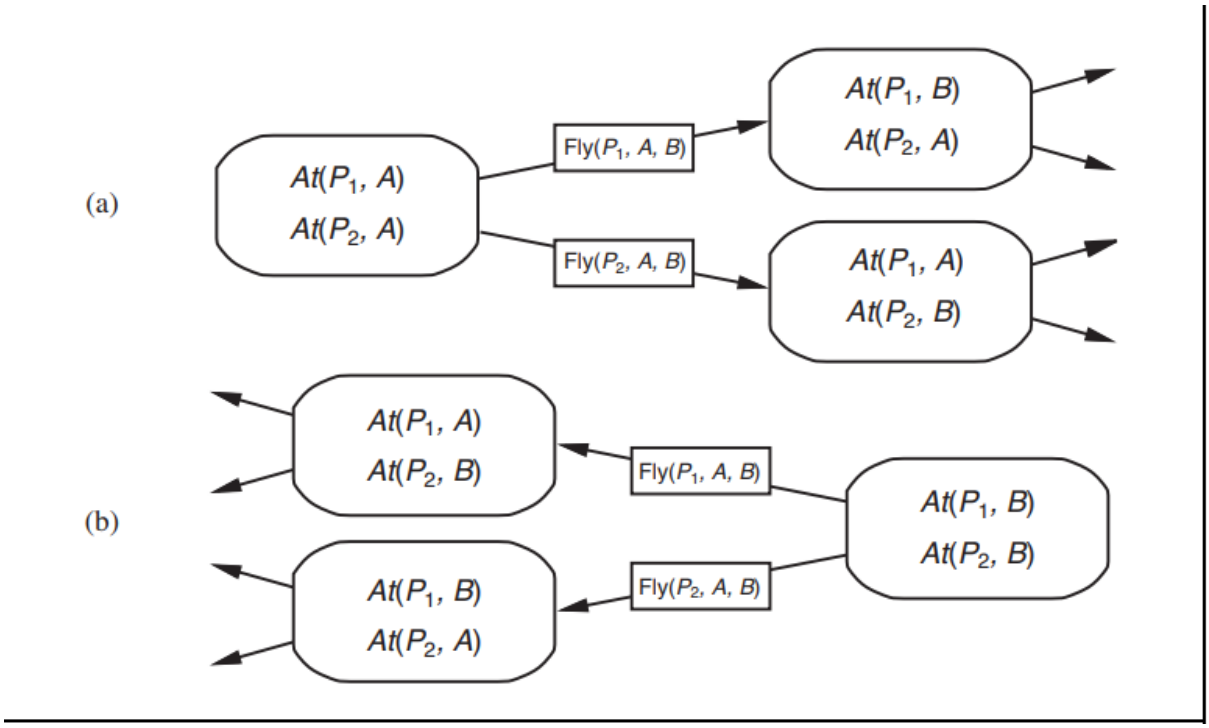


Figure 10.4 Diagram of the blocks-world problem in Figure 10.3.



PLANNING AND ACTING IN NONDETERMINISTIC DOMAINS

In this section we extend planning to handle partially observable, nondeterministic, and unknown environments. Chapter 4 extended search in similar ways, and the methods here are also similar: sensorless planning (also known as conformant planning) for environments with no observations; contingency planning for partially observable and nondeterministic environments; and online planning and replanning for unknown environments.

$Action(RemoveLid(can),$
PRECOND: $Can(can)$
EFFECT: $Open(can))$
$Action(Paint(x, can),$
PRECOND: $Object(x) \wedge Can(can) \wedge Color(can, c) \wedge Open(can)$
EFFECT: $Color(x, c))$

13.1 ACTING UNDER UNCERTAINTY

UNCERTAINTY

Agents may need to handle **uncertainty**, whether due to partial observability, nondeterminism, or a combination of the two. An agent may never know for certain what state it's in or where it will end up after a sequence of actions.

We have seen problem-solving agents (Chapter 4) and logical agents (Chapters 7 and 11) designed to handle uncertainty by keeping track of a **belief state**—a representation of the set of all possible world states that it might be in—and generating a contingency plan that handles every possible eventuality that its sensors may report during execution. Despite its many virtues, however, this approach has significant drawbacks when taken literally as a recipe for creating agent programs:

- When interpreting partial sensor information, a logical agent must consider *every logically possible* explanation for the observations, no matter how unlikely. This leads to impossible large and complex belief-state representations.
- A correct contingent plan that handles every eventuality can grow arbitrarily large and must consider arbitrarily unlikely contingencies.
- Sometimes there is no plan that is guaranteed to achieve the goal—yet the agent must act. It must have some way to compare the merits of plans that are not guaranteed.

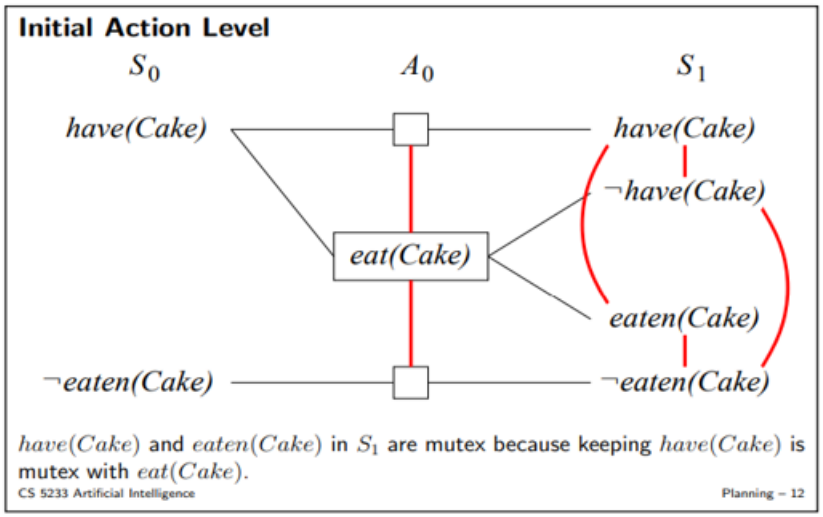
1 **Summarizing uncertainty**

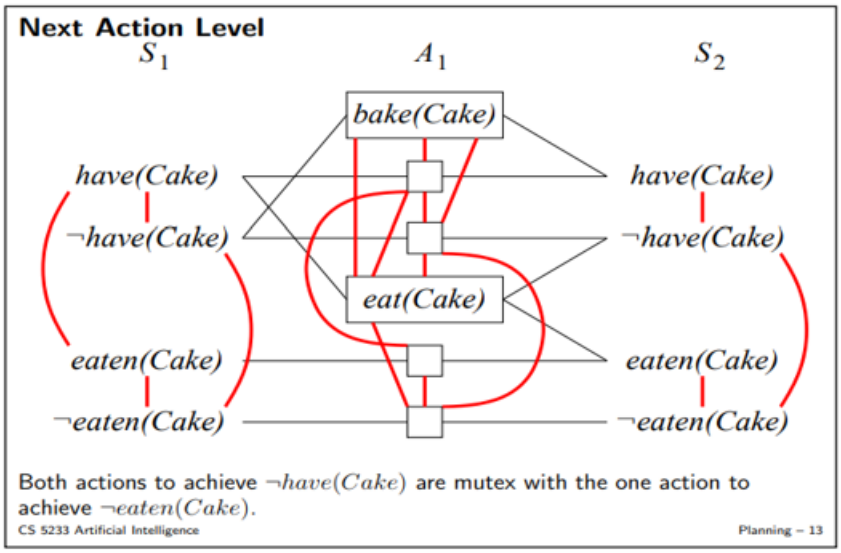
Let's consider an example of uncertain reasoning: diagnosing a dental patient's toothache. Diagnosis—whether for medicine, automobile repair, or whatever—almost always involves uncertainty. Let us try to write rules for dental diagnosis using propositional logic, so that we can see how the logical approach breaks down.

Consider the following simple rule: Toothache ⇒ Cavity .

The problem is that this rule is wrong. Not all patients with toothaches have cavities; some of them have gum disease, an abscess, or one of several other problems:

Toothache ⇒ Cavity ∨ GumProblem ∨ Abscess ...





OTHER APPROACHES TO UNCERTAIN REASONING

Other sciences (e.g., physics, genetics, and economics) have long favored probability as a model for uncertainty. In 1819, Pierre Laplace said, “Probability theory is nothing but common sense reduced to calculation.” In 1850, James Maxwell said, “The true logic for this world is the calculus of Probabilities, which takes account of the magnitude of the probability which is, or ought to be, in a reasonable man’s mind.”

14.7.1 Rule-based methods for uncertain reasoning

Rule-based systems emerged from early work on practical and intuitive systems for logical inference. Logical systems in general, and logical rule-based systems in particular, have three desirable properties:

LOCALITY

- **Locality**: In logical systems, whenever we have a rule of the form $A \Rightarrow B$, we can conclude B, given evidence A, *without worrying about any other rules.* In probabilistic systems, we need to consider *all* the evidence.

DETACHMENT

- **Detachment**: Once a logical proof is found for a proposition B, the proposition can be used regardless of how it was derived. That is, it can be **detached** from its justification. In dealing with probabilities, on the other hand, the source of the evidence for a belief is important for subsequent reasoning.

TRUTH-FUNCTIONALITY

- **Truth-functionality**: In logic, the truth of complex sentences can be computed from the truth of the components. Probability combination does not work this way, except under strong global independence assumptions.

TIME AND UNCERTAINTY

We have developed our techniques for probabilistic reasoning in the context of static worlds, in which each random variable has a single fixed value. For example, when repairing a car, we assume that whatever is broken remains broken during the process of diagnosis; our job is to infer the state of the car from observed evidence, which also remains fixed.

Acting under Uncertainty

Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates. With this knowledge representation, we might write A→B, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.

So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

Causes of uncertainty:

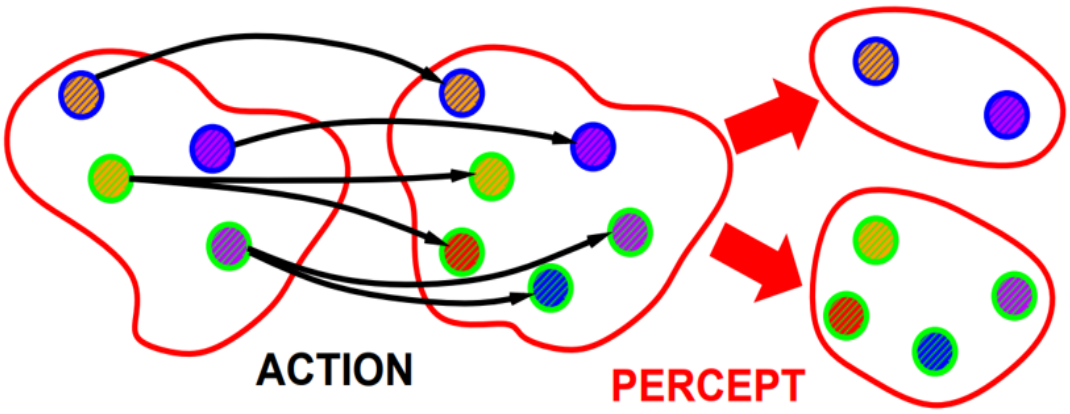
Following are some leading causes of uncertainty to occur in the real world.

1. Information occurred from unreliable sources.
2. Experimental Errors
3. Equipment fault
4. Temperature variation
5. Climate change.

Probabilistic reasoning:

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.

We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.



Need of probabilistic reasoning in AI:

- When there are unpredictable outcomes.
- When specifications or possibilities of predicates becomes too large to handle.
- When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:

- **Bayes' rule**
- **Bayesian Statistics**