action that results from substituting values for all the variables:

\[ Action(Fly(P_i, SFO, JFK), \]
\[
\text{PRECOND: } \text{At}(P_i, SFO) \land \text{Plane}(P_i) \land \text{Airport}(SFO) \land \text{Airport}(JFK) \]
\[
\text{EFFECT: } \neg \text{At}(P_i, SFO) \land \text{At}(P_i, JFK) \]

The precondition and effect of an action are each conjunctions of literals (positive or negated atomic sentences). The precondition defines the states in which the action can be executed, and the effect defines the result of executing the action. An action \( a \) can be executed in state \( s \) if \( s \) entails the precondition of \( a \). Entailment can also be expressed with the set semantics:

\( q \iff \text{every positive literal in } q \text{ is in } s \text{ and every negated literal in } q \text{ is not.} \)

In formal notation we say

\( (a \in ACTIONS(s)) \land S = \text{PRECOND}(a) \),

where any variables in \( a \) are universally quantified. For example,

\( V p, from, to \) to \( \{Fly(p, from, to) \in ACTIONS(s)\} \Rightarrow (\text{At}(p, from) \land \text{Plane}(p) \land \text{Airport}(from) \land \text{Airport}(to)) \)

We say that action \( a \) is applicable in state \( s \) if the preconditions are satisfied by \( s \). When an action schema \( a \) contains variables, it may have multiple applicable instantiations. For example, with the initial state defined in Figure 10.1, the \( Fly \) action can be instantiated as \( Fly(P_1, JFK, SFO) \) or as \( Fly(P_2, JFK, SFO) \), both of which are applicable in the initial state. If an action \( a \) has \( v \) variables, then, in a domain with \( k \) unique names of objects, it takes \( OW \) time in the worst case to find the applicable ground actions.

Sometimes we want to propositionalize a PDDL problem—replace each action schema with a set of ground actions and then use a propositional solver such as SATPLAN to find a solution. However, this is impractical when \( v \) and \( k \) are large.

The result of executing action \( a \) in state \( s \) is defined as a state \( s' \) which is represented by the set of fluents formed by starting with \( s \), removing the fluents that appear as negative literals in the action’s effects (what we call the delete list or \( \text{DEL}(a) \)), and adding, the fluent that are positive literals in the action’s effects (what we call the add list or \( \text{ADD}(a) \)):

\[
\text{RESULT}(s, a) = (s - \text{DEL}(a)) \cup \text{ADD}(a) .
\]

For example, with the action \( Fly(P_1, SFO, JFK) \), we would remove \( \text{At}(P_1, SFO) \) and add \( \text{At}(P_1, JFK) \). It is a requirement of action schemas that any variable in the effect must also appear in the precondition. That way, when the precondition is matched against the state \( s \), all the variables will be bound, and \( \text{RESULT}(s, a) \) will therefore have only ground atoms. In other words, ground states are closed under the \( \text{RESULT} \) operation.

Also note that the fluents do not explicitly refer to time, as they did in Chapter 7. There we needed superscripts for time, and successor-state axioms of the form

\[
\text{ActionCauses} F \lor (F^t \land \text{ActionCausesNot} F) .
\]

In PDDL the times and states are implicit in the action schemas: the precondition always refers to time \( t \) and the effect to time \( t + 1 \).

A set of action schemas serves as a definition of a planning domain. A specific problem within the domain is defined with the addition of an initial state and a goal. The initial
Section 10.1. Definition of Classical Planning

**Practice Problem:**

Consider the following problem specification:

- **Objects:**
  - Airports: JFK, SFO
  - Planes: Plane1, Plane2
  - Cargo: Cargo1, Cargo2

- **Predicates:**
  - **At:** An object is at a location.
  - **In:** An object is inside a plane.
  - **Load:** Load cargo into a plane.
  - **Unload:** Unload cargo from a plane.
  - **Plane:** State of a plane.
  - **Cargo:** State of cargo.

- **Initial State:**
  - At(Cargo1, JFK), In(Cargo1, Plane1)
  - At(Cargo2, SFO), In(Cargo2, Plane2)

- **Goal:**
  - At(Cargo1, SFO)

**Question:** Formulate the problem using PDDL.

**Solution:**

```pddl
Init(At(Cargo1, JFK) ∧ At(Cargo2, SFO) ∧ Plane(Plane1) ∧ Plane(Plane2) ∧ 
     Airport(JFK) ∧ Airport(SFO))

Goal(At(Cargo1, SFO))
```

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

---

**State** is a conjunction of ground atoms. (As with all states, the closed-world assumption is used, which means that any atoms that are not mentioned are false.) The goal is just like a precondition: a conjunction of literals (positive or negative) that may contain variables, such as `At(p, SFO) ∧ Plane(p)`. Any variables are treated as existentially quantified, so this goal is to have any plane at SFO. The problem is solved when we can find a sequence of actions that end in a state that entails the goal. For example, the state `Rich ∧ Famous ∧ Miserable` entails the goal `Rich ∧ Famous`, and the state `Plane(Plane1) ∧ At(Plane1, SFO)` entails the goal `At(p, SFO) ∧ Plane(p)`.

Now we have defined planning as a search problem: we have an initial state, an ACTIONS function, a RESULT function, and a goal test. We’ll look at some example problems before investigating efficient search algorithms.

**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(Cargo1, Plane1, SFO), Unload(Cargo2, Plane2, JFK), 
Load(Cargo2, Plane2, JFK), Fly(Plane1, JFK, SFO), Unload(Cargo1, Plane1, SFO).
```