Planning

CS156

Chris Pollett

Nov 3, 2014

Outline

- Planning, PDDL
- Planning Examples
- Forward and Back Search

Introduction

- Last day, we finished up talking about First-order Logic and theorem-proving algorithms in first-order logic using a combination of resolution and unification.
- Today, we are going to use first-order logic to express planning problems and develop logical agents which can solve them.
- Recall we mentioned planning agents when talking about goal-based agents: We said goal-based agents restricted to atomic domains are called problem-solving agents and goal-based agents on factored domains where called planning-agents.

Planning, PDDL

- To start we look at some specific logics for expressing planning problems...
- The main logic we are going to consider is PDDL (Planning Domain Definition Language).
- PDDL was developed by Drew McDermott in 1998 and was heavily influenced by earlier planning languages such as STRIPS and ADL.
- One reason for its development was so that researcher could have a common planning language for International Planning Competition.
- PDDL has also served as the basis for several notable extended languages that can handle multi-agent (MA-PDDL, MAPL), probabilistic (PPDDL), and temporal environments (NASA's NDDL).
- As of Fall 2014, the most recent version of PDDL is Version 3.1.

PDDL Fluents, States

- As planning is a kind of search problem, we need to define the initial state, the set of actions available in a state, the results of applying an action, and a goal test
- Each **state** is represented as a conjunction of **fluents** (facts that might change from situation to situation) that are ground (slots filled by terms not involving variables), functionless atoms. For example, $Poor \wedge Unknown$ might represent the state of a hapless agent. An example state from a package delivery problem might be $At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$.
- In our system we make use of the **Closed World Assumption (CWA)**. If a fluent is not in our KB then it is false.
- We also make use of the **unique names assumption**: So if we mention $Truck_1$ and $Truck_2$ we assume $Truck_1 \neq Truck_2$.
- The following fluents would not be in a state description: $At(x, y)$, as it is not ground as x and y are variable); $\neg Poor$ as it is a negation of an atom, and $At(Father(Fred), Sydney)$, as it uses functions.
- Since it is a conjunction, the order of writing the fluents does not matter and we can also use sets of fluents to represent states.

PDDL Actions

- To define Actions, we make use of a schemas that implicitly define the ACTIONS(s) and RESULT(s,a) functions needed for a problem-solving search.
- One thing that we require of our schemas is that they clearly define what changes and what stays the same as a result of an action.
- Otherwise, it is often quite hard to calculate what stays the same. This is known as the **frame problem**.
- As classical planning concerns itself with problems whose actions leave most things in the state unchanged, PDDL specifies the result of an action by only mentioning what changes, everything that stays the same is unmentioned.
- An example action in PDDL might be:

$$\begin{array}{l} Action(Fly(p, from, to), \\ \quad Precond: At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to), \\ \quad Effect: \neg At(p, from) \wedge At(p, to) \quad) \end{array}$$

- A schema consists of the action name, a list of all the variables used in the schema, a **precondition**, and an **effect**.
- We are free to choose whatever values we want to instantiate the variables.

PDDL Actions Example

- A particular instance, of the last action might be:

  $$\begin{array}{l} Action(Fly(P_1, SFO, JFK), \\ \quad Precond: At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK), \\ \quad Effect: \neg At(P_1, SFO) \wedge At(P_1, JFK) \quad) \end{array}$$

- Entailment can be expressed with the set semantics $s \models q$ iff every positive literal in q is in s and every negated literal in q is not. In formal notation we say:
  $(a \in Action(s)) \Leftrightarrow s \models Precond(a)$
  Here any variables in a are universally quantified.
- We say that an action a is **applicable** in state s if the preconditions are satisfied by s.
- Different instantiations of the same action can be applicable in a given state. For example, in the initial state, $Fly(P_1, SFO, JFK)$ and $Fly(P_2, JFK, SFO)$ might both be applicable.
- If an action has v variables, then in a domain with k unique names of objects, it takes $O(k^v)$ time in the worst case (try each setting of each variable) to find the applicable ground actions.

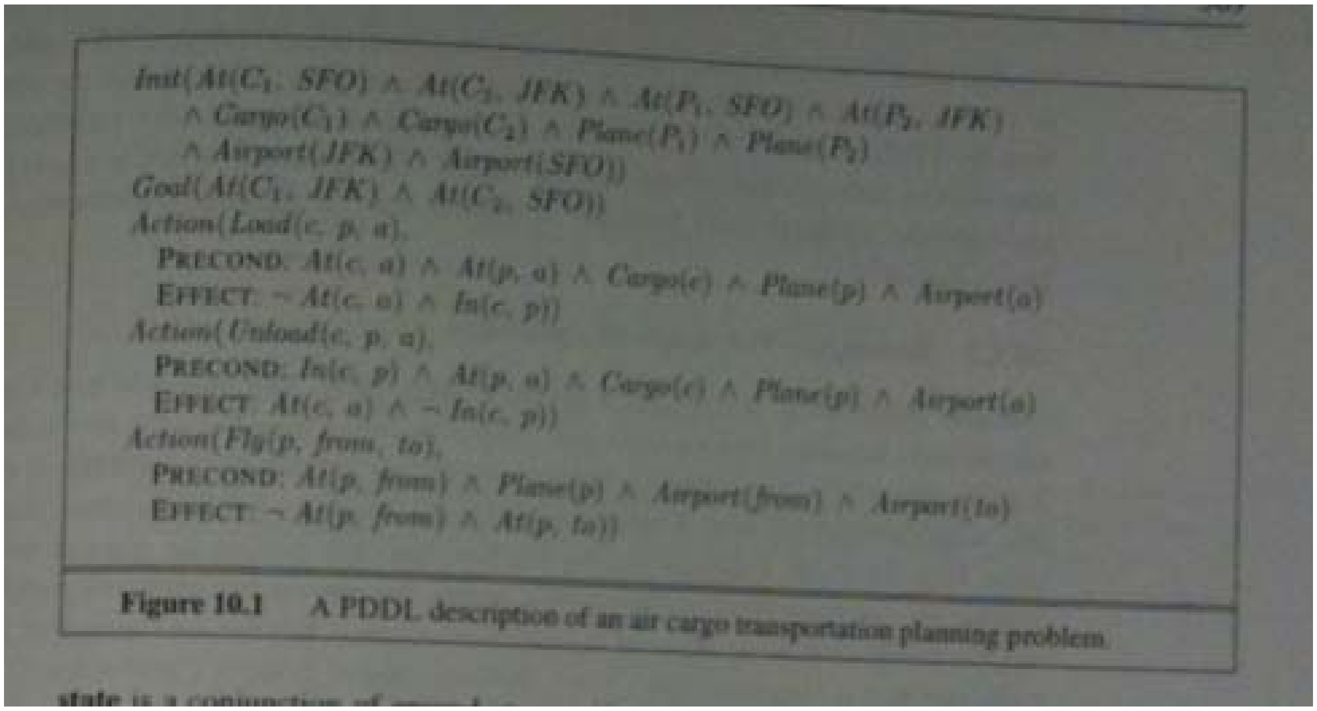
PDDL Results of Action

- The **result** of executing action a in state s consists in removing the fluents that appear as negative literals in the action's effects (the **delete list**, $Del(a)$), and adding the fluents that are positive literals in the action's effects (the **add list** or $Add(a)$):
  $Result(s, a) = (s - Del(a)) \cup Add(a)$.
- For example, with the action $Fly(P_1, SFO, JFK)$, we would remove $At(P_1, SFO)$ and add $At(P_1, JFK)$.
- We require that any variable in the effect of an action schema be also in the precondition.
- This ensures that when we bind the precondition to ground atoms, the result will also contain only ground atoms and no variables.
- Notice that fluents do not explicitly refer to time: Preconditions always refer to time t and Effects always refer to time $t+1$.

PDDL - A Complete Problem

- A set of action schemas serves as a definition of a **planning domain**.
- A specific problem is defined with the addition of an initial state and a goal.
- As we said before, the **initial state** is a conjunction of ground atoms.
- A **goal** is a conjunction of literals.
- Variables in this goal are treated as existentially quantified.
- So to satisfy a goal like $At(p, SFO) \wedge Plane(p)$ it suffices to have *any* plane at SFO.
- A planning problem is **solved** when we can find a sequence of actions that end in a state s that entails the goal.
- So for example, $Rich \wedge Famous \wedge Miserable$ entails $Rich \wedge Famous$; the state $Plane(P_1) \wedge At(P_1, SFO)$ entails $At(p, SFO) \wedge Plane(p)$.
- So now we have completely defined a search problem, we have an initial state, an Actions function, a Result function, and a goal test... Let's look at an example.

Example Planning Problem



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

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

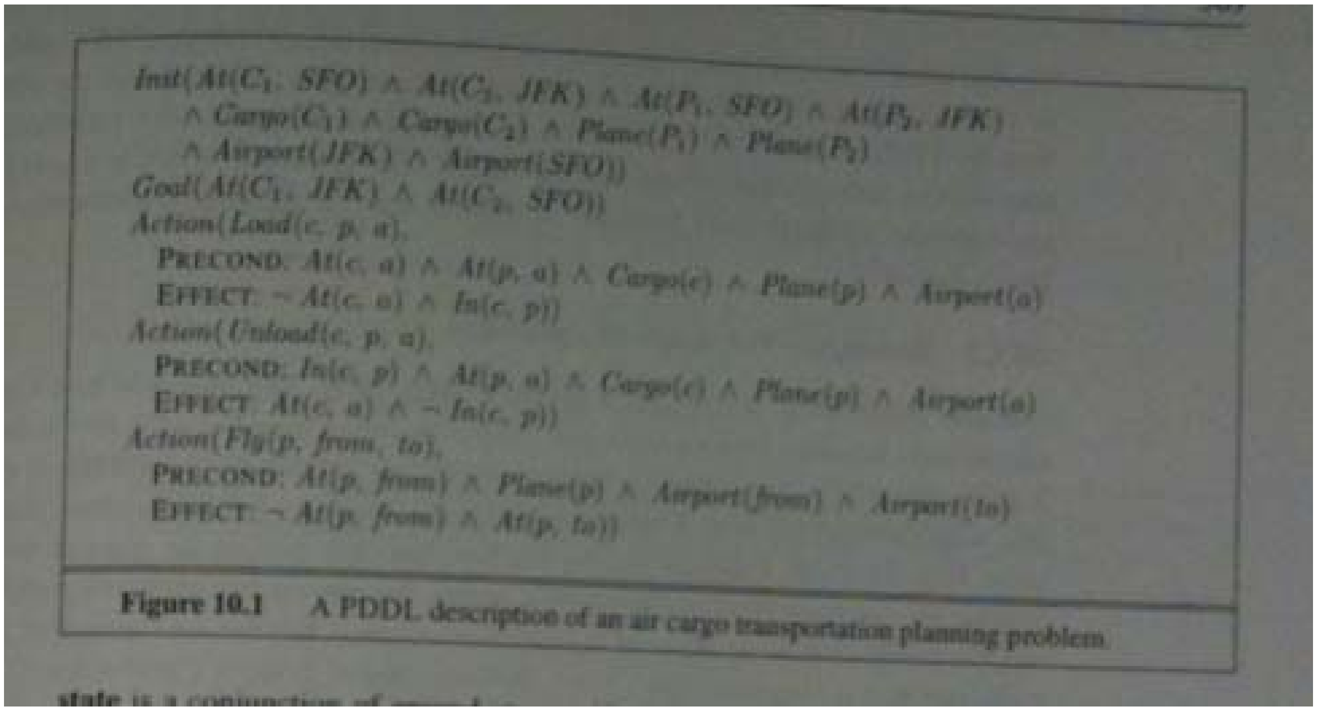
state is a conjunction of

Quiz

Which of the following is true?

1. A given first-order formula is true in at most one model.
2. The theorem proving algorithm suggested last week involved introducing new function symbols for forall there exists blocks.
3. $A(t) \Rightarrow \forall x A(x)$ where t is a term is logically valid.

Cargo Problem

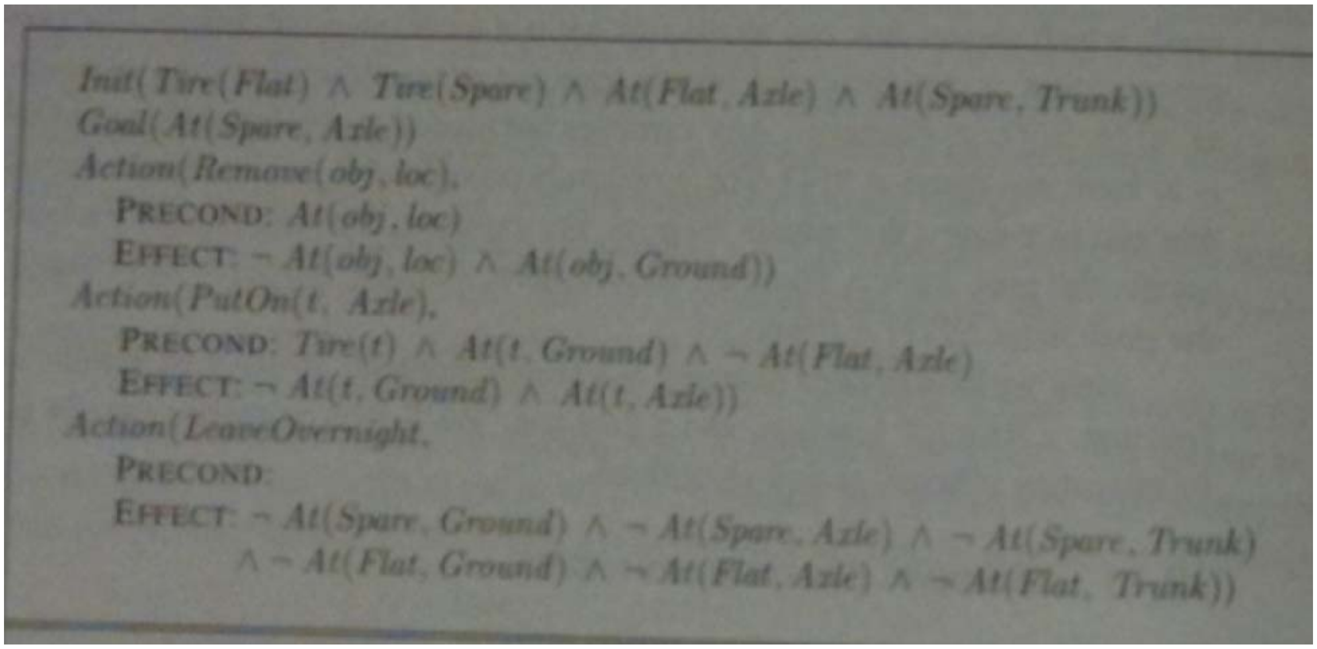


```
Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(P1, SFO) ∧ At(P2, JFK)
    ∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2)
    ∧ Airport(JFK) ∧ Airport(SFO))
Goal(At(C1, JFK) ∧ At(C2, SFO))
Action(Load(c, p, a),
  PRECOND: At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
  EFFECT: ¬ At(c, a) ∧ In(c, p))
Action(Unload(c, p, a),
  PRECOND: In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
  EFFECT: At(c, a) ∧ ¬ In(c, p))
Action(Fly(p, from, to),
  PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
  EFFECT: ¬ At(p, from) ∧ At(p, to))
```

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

- An example, solution to this problem is the sequence:
  $[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK),$
  $Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)]$
- This problem illustrates one thing that we will have to be careful about when we talk about algorithms...
- Consider the action $Fly(P_1, JFK, JFK)$. Intuitively, this should do nothing. However, the effect according to the fly rule is $At(P_1, JFK) \wedge \neg At(P_1, JFK)$. Our rules for changing from one state say that we add $At(P_1, JFK)$ (no change to the state set) then delete $At(P_1, JFK)$, causing the plane to vanish!
- Typically, this problem is solved by adding inequality condiitons on the from and to airports.

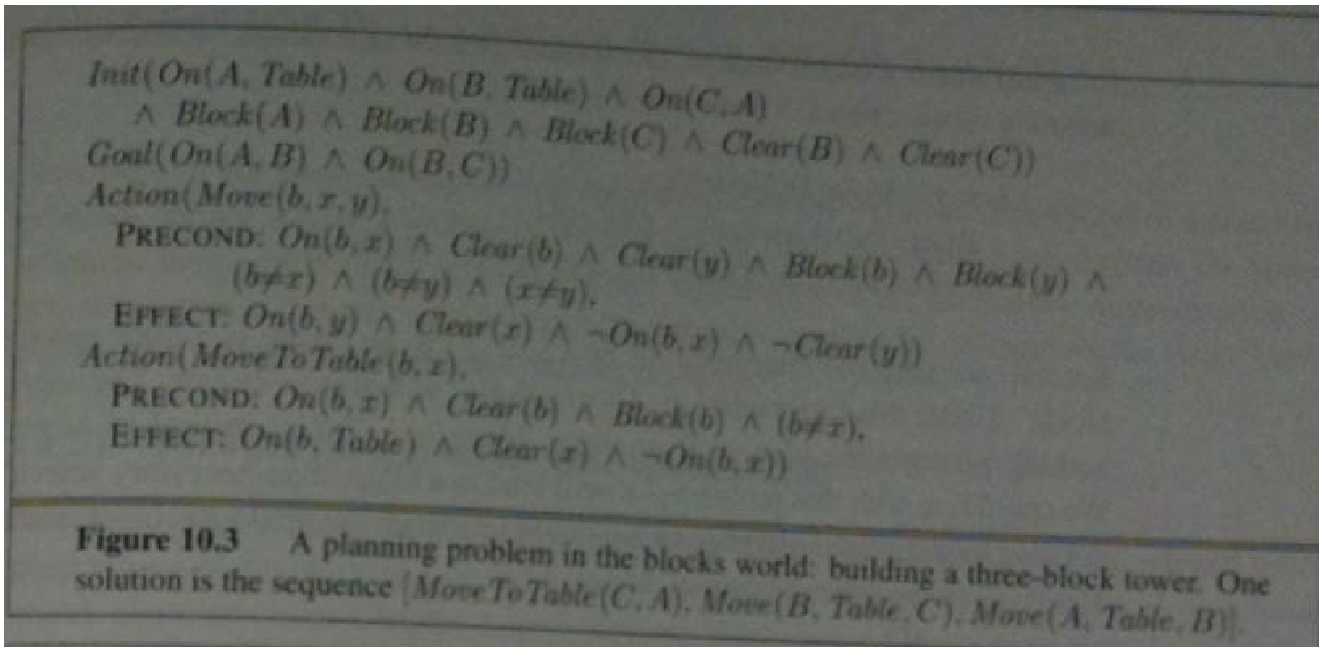
The Spare Tire Problem



```
Init(Tire(Flat) ∧ Tire(Spare) ∧ At(Flat, Axle) ∧ At(Spare, Trunk))
Goal(At(Spare, Axle))
Action(Remove(obj, loc),
   PRECOND: At(obj, loc)
   EFFECT: ¬ At(obj, loc) ∧ At(obj, Ground))
Action(PutOn(t, Axle),
   PRECOND: Tire(t) ∧ At(t, Ground) ∧ ¬ At(Flat, Axle)
   EFFECT: ¬ At(t, Ground) ∧ At(t, Axle))
Action(LeaveOvernight,
   PRECOND:
   EFFECT: ¬ At(Spare, Ground) ∧ ¬ At(Spare, Axle) ∧ ¬ At(Spare, Trunk)
           ∧ ¬ At(Flat, Ground) ∧ ¬ At(Flat, Axle) ∧ ¬ At(Flat, Trunk))
```

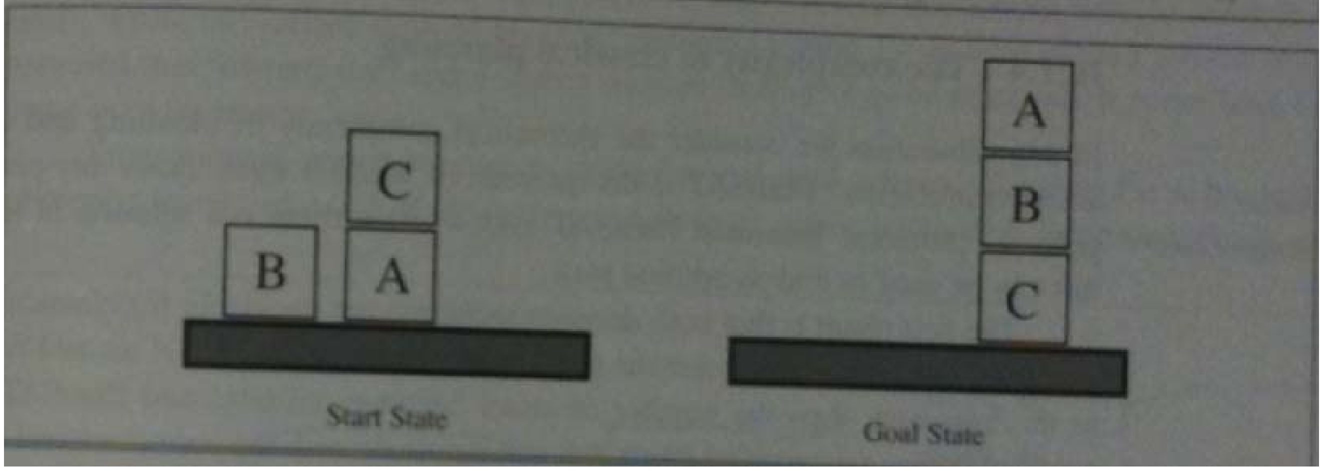
- Consider the problem of changing a flat tire. This can be expressed using the problem statement above from the book.
- The goal is to have a good spare tire properly mounted onto the car's axle. The initial state has a flat tire on the axle and a good spare in the trunk.
- A solution to the problem is the sequence: $[Remove(Flat, Axle), Remove(Spare, Trunk), PutOn(Spare, Axle)]$

Blocks World



$Init(On(A, Table) \wedge On(B, Table) \wedge On(C, A)$
$\quad \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(B) \wedge Clear(C))$
$Goal(On(A, B) \wedge On(B, C))$
$Action(Move(b, x, y),$
$\quad$ PRECOND: $On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge Block(y) \wedge$
$\qquad (b \neq x) \wedge (b \neq y) \wedge (x \neq y),$
$\quad$ EFFECT: $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$
$Action(MoveToTable(b, x),$
$\quad$ PRECOND: $On(b, x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x),$
$\quad$ EFFECT: $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$

Figure 10.3 A planning problem in the blocks world: building a three-block tower. One solution is the sequence $[MoveToTable(C, A), Move(B, Table, C), Move(A, Table, B)]$.



- One of the most famous early AI planners was SHRDLU (1968-1970) which allowed a human to converse with a program about objects in block world. Using English the human could instruct the program to rearrange, stack blocks, etc, and the program would figure out the sequence of actions needed to accomplish this.
- Above we give a simplified block world, state it as a planning domain, and give an initial and goal state.
- Here $On(b, x)$ indicates that block b is on x, where x is another block or the table. $Move(b, x, y)$ moves block b from on top of x to on top of y provided that nothing is on top of b, b is on top of x and nothing is on top of y.
- In order to allow the table to hold multiple blocks, we have a separate action for it. We also in the above interpret $Clear(x)$ to be there is a clear space on x to hold a block.

The Complexity of Classical Planning

- Two common questions associated with planning are:
  - ***PlanSat*** *- given a planning problem, does there exist any plan that solves the problem?*
  - ***BoundedPlanSat*** *- given a planning problem, does there exist any plan that solves the problem in k steps of less?*
- Both of these problems are known to be decidable for classical planning, that is, there is an algorithm (maybe inefficient) to find a plan if it exists.
- Further (Ghallab 2004) shows that both problems are in fact in PSPACE but are NP-hard.
- They remain NP-hard if we disallow negative effects from our actions. However, PlanSat, for problems without negative preconditions is in polynomial time (P).
- These are worst-case results. Many real-world problems are much easier than the worst case. But these results do show that there is an art to solving planning problems, and this art often means coming up with good heuristics.

Forward State-Space Search

- In **forward search**, we start with our initial state and we apply each possible action rule and search forward hoping to find a goal state.
- One problem with forward search is it is prone to considering irrelevant actions.
- For example, suppose the goal is to buy a textbook and we have an action schema $Buy(isbn)$ with effect $Own(isbn)$. ISBNs are ten digits, so this action schema has 10 billion ground actions. An uninformed forward-search would have to enumerate through each.
- Planning problems often have large state spaces. Consider an air cargo problem with 10 airports, where each airport has 5 planes and 20 pieces of cargo. The goal is to move all the cargo at airport A to airport B. This could be done by loading 20 pieces of cargo at airport A into one plane, fly the plane to B, and unload the cargo.
- Finding the solution, however, might be hard because each of the 50 planes can fly to 9 other airports, and each of the 200 packages can be loaded or unloaded into any planes at its airport. The book estimates this problem might have as many as 2000^{41} possible states.
- For forward planning to work we need to have good domain-independent heuristics (which we will see we do).

Backward (Regression) Relevant-States Search

- In **backward (regression) search**, we start from the goal, and try to search backward until we reach a state implied by the initial state.
- This kind of search is also called **relevant-states** search because we only consider actions that are relevant to the goal.
- At each step, there will be a set of **relevant states** to consider, not just a single state.
- For example, suppose the goal was $\neg Poor \wedge Famous$. This describes those states in which Poor is false, Famous is true, and we don't care about the value of the other states.
- If there are n ground fluents in a domain, then there are 2^n ground states, but 3^n descriptions of sets of goals: positive, negative, and not mentioned.
- If a domain is expressed in PDDL, it is not too hard to do regression on it. Given a state g, action a (everything ground), the regression from g to a gives us a state g' defined by
  $g' = (g - ADD(a)) \cup Precond(a)$
- In addition to being able to handle ground states, we need to be able to handle partially un-instantiated actions and states.
- For example, suppose the goal is to deliver a specific piece of cargo to SFO: $At(C_2, SFO)$. This suggests the action $Unload(C_2, p', SFO)$:
  $Action(Unload(C_2, p', SFO),$
  $Precond: In(C_2, p') \wedge At(p', SFO) \wedge Cargo(C_2) \wedge Plane(p') \wedge Airport(SFO)$,
  $Effect: At(C_2, SFO) \wedge \neg In(C_2, p'))$.