EXERCISES 11

AUTOMATED PLANNING

11.1 Definition of Classical Planning

Exercise 11.PSPL

Describe the differences and similarities between problem solving and planning.

Both problem solver and planner are concerned with getting from a start state to a goal using a set of defined operations or actions, most commonly in a deterministic, discrete, observable environment (although those constraints can be relaxed). In problem solving we use atomic representations. In planning, however, we open up the representation of states, goals, and plans, using factored or relational representations that allow for a wider variety of algorithms that decompose the search space, search forwards or backwards, and use automated generation of heuristic functions.

Exercise 11.PDDR

Consider a robot whose operation is described by the following PDDL operators:

$$Op(\textsc{Action}{:}Go(x,y), \textsc{Precond}{:}At(Robot,x), \textsc{Effect}{:}\neg At(Robot,x) \land At(Robot,y))$$
$$Op(\textsc{Action}{:}Pick(o), \textsc{Precond}{:}At(Robot,x) \land At(o,x), \textsc{Effect}{:}\neg At(o,x) \land Holding(o))$$
$$Op(\textsc{Action}{:}Drop(o), \textsc{Precond}{:}At(Robot,x) \land Holding(o), \textsc{Effect}{:}At(o,x) \land \neg Holding(o))$$

a. The operators allow the robot to hold more than one object. Show how to modify them with an $EmptyHand$ predicate for a robot that can hold only one object.

b. Assuming that these are the only actions in the world, write a successor-state axiom for $EmptyHand$.

a. $EmptyHand$ is not affected by Go, so we modify just the $Pick$ and $Drop$ operators:

$$\begin{aligned}&Op(\textsc{Action}{:}Pick(o), \textsc{Precond}{:}EmptyHand() \land At(Robot,x) \land At(o,x),\\ &\quad \textsc{Effect}{:}\neg EmptyHand() \land \neg At(o,x) \land Holding(o))\\ &Op(\textsc{Action}{:}Drop(o), \textsc{Precond}{:}At(Robot,x) \land Holding(o),\\ &\quad \textsc{Effect}{:}EmptyHand() \land At(o,x) \land \neg Holding(o))\end{aligned}$$

Notice that STRIPS does not allow negated preconditions, so we could not use $\neg Holding(p)$ as a precondition for $Pick(o)$; this is why we need $EmptyHand$. Also, we cannot use

$\neg EmptyHand$ as a precondition of $Drop(o)$; but this is no problem because we already have $Holding(o)$ as a precondition.

b. In English, the hand is empty after doing an action if it was empty before and the action was not a successful $Pick$; or if an object was dropped.

$$\begin{aligned}&EmptyHand(Result(a,s)) \Leftrightarrow\\&\quad[(EmptyHand(s) \land \neg\exists\, o,x\ \ (At(Robot,x) \land At(o,x) \land a = Pick(o)))\\&\quad\lor (Holding(o,s) \land a = Drop(o))]\end{aligned}$$

Exercise 11.ARST

Given the action schemas and initial state from Figure 11.1, what are all the applicable concrete instances of $Fly(p, from, to)$ in the state described by

$$\begin{aligned}&At(P_1, JFK) \land At(P_2, SFO) \land Plane(P_1) \land Plane(P_2)\\&\quad\land Airport(JFK) \land Airport(SFO)\ ?\end{aligned}$$

This is an easy exercise, the point of which is to understand that "applicable" means satisfying the preconditions, and that a concrete action instance is one with the variables replaced by constants. The applicable actions are:

$$\begin{aligned}&Fly(P_1, JFK, SFO)\\&Fly(P_1, JFK, JFK)\\&Fly(P_2, SFO, JFK)\\&Fly(P_2, SFO, SFO)\end{aligned}$$

A minor point of this is that the action of flying nowhere—from one airport to itself—is allowable by the definition of Fly, and is applicable (if not useful).

Exercise 11.MOBA

The monkey-and-bananas problem is faced by a monkey in a laboratory with some bananas hanging out of reach from the ceiling. A box is available that will enable the monkey to reach the bananas if he climbs on it. Initially, the monkey is at A, the bananas at B, and the box at C. The monkey and box have height Low, but if the monkey climbs onto the box he will have height $High$, the same as the bananas. The actions available to the monkey include Go from one place to another, $Push$ an object from one place to another, $ClimbUp$ onto or $ClimbDown$ from an object, and $Grasp$ or $Ungrasp$ an object. The result of a $Grasp$ is that the monkey holds the object if the monkey and object are in the same place at the same height.

a. Write down the initial state description.

b. Write the six action schemas.

c. Suppose the monkey wants to fool the scientists, who are off to tea, by grabbing the bananas, but leaving the box in its original place. Write this as a general goal (i.e., not assuming that the box is necessarily at C) in the language of situation calculus. Can this goal be solved by a classical planning system?

d. Your schema for pushing is probably incorrect, because if the object is too heavy, its position will remain the same when the *Push* schema is applied. Fix your action schema to account for heavy objects.

This exercise is intended as a fairly easy exercise in describing a domain.

a. The initial state is:

$$\begin{array}{l} At(Monkey, A) \land At(Bananas, B) \land At(Box, C) \land \\ Height(Monkey, Low) \land Height(Box, Low) \land Height(Bananas, High) \land \\ Pushable(Box) \land Climbable(Box) \end{array}$$

b. The actions are:

$$\begin{array}{l} Action(\textsc{Action}{:}Go(x,y), \textsc{Precond}{:}At(Monkey,x), \\ \quad \textsc{Effect}{:}At(Monkey,y) \land \neg(At(Monkey,x))) \\ Action(\textsc{Action}{:}Push(b,x,y), \textsc{Precond}{:}At(Monkey,x) \land Pushable(b), \\ \quad \textsc{Effect}{:}At(b,y) \land At(Monkey,y) \land \neg At(b,x) \land \neg At(Monkey,x)) \\ Action(\textsc{Action}{:}ClimbUp(b), \\ \quad \textsc{Precond}{:}At(Monkey,x) \land At(b,x) \land Climbable(b), \\ \quad \textsc{Effect}{:}On(Monkey,b) \land \neg Height(Monkey,Low) \\ \qquad \land Height(Monkey,High) \\ Action(\textsc{Action}{:}Grasp(b), \\ \quad \textsc{Precond}{:}Height(Monkey,h) \land Height(b,h) \\ \qquad \land At(Monkey,x) \land At(b,x), \\ \quad \textsc{Effect}{:}Have(Monkey,b)) \\ Action(\textsc{Action}{:}ClimbDown(b), \\ \quad \textsc{Precond}{:}On(Monkey,b) \land Height(Monkey,High), \\ \quad \textsc{Effect}{:}\neg On(Monkey,b) \land \neg Height(Monkey,High) \\ \qquad \land Height(Monkey,Low) \\ Action(\textsc{Action}{:}UnGrasp(b), \textsc{Precond}{:}Have(Monkey,b), \\ \quad \textsc{Effect}{:}\neg Have(Monkey,b)) \end{array}$$

c. In situation calculus, the goal is a state s such that:

$$Have(Monkey, Bananas, s) \land (\exists x \;\; At(Box, x, s_0) \land At(Box, x, s))$$

In STRIPS, we can only talk about the goal state; there is no way of representing the fact that there must be some relation (such as equality of location of an object) between two states within the plan. So there is no way to represent this goal.

d. Actually, we did include the *Pushable* precondition in the solution above.

Exercise 11.SHAK

The original STRIPS planner was designed to control Shakey the robot. Figure 11.1 shows a version of Shakey's world consisting of four rooms lined up along a corridor, where each room has a door and a light switch. The actions in Shakey's world include moving from place to place, pushing movable objects (such as boxes), climbing onto and down from rigid objects (such as boxes), and turning light switches on and off. The robot itself could not climb on a box or toggle a switch, but the planner was capable of finding and printing out plans that were beyond the robot's abilities. Shakey's six actions are the following:

- $Go(x, y, r)$, which requires that Shakey be At x and that x and y are locations In the same room r. By convention a door between two rooms is in both of them.
- Push a box b from location x to location y within the same room: $Push(b, x, y, r)$. You will need the predicate Box and constants for the boxes.
- Climb onto a box from position x: $ClimbUp(x, b)$; climb down from a box to position x: $ClimbDown(b, x)$. We will need the predicate On and the constant $Floor$.
- Turn a light switch on or off: $TurnOn(s, b)$; $TurnOff(s, b)$. To turn a light on or off, Shakey must be on top of a box at the light switch's location.

Write PDDL sentences for Shakey's six actions and the initial state from Figure 11.1. Construct a plan for Shakey to get Box_2 into $Room_2$.

The actions are quite similar to the monkey and bananas problem—you should probably assign only one of these two problems. The actions are:

$$\begin{aligned}
&Action(\text{ACTION:}Go(x, y), \text{PRECOND:}At(Shakey, x) \land In(x, r) \land In(y, r),\\
&\quad \text{EFFECT:}At(Shakey, y) \land \neg(At(Shakey, x)))\\
&Action(\text{ACTION:}Push(b, x, y), \text{PRECOND:}At(Shakey, x) \land Pushable(b),\\
&\quad \text{EFFECT:}At(b, y) \land At(Shakey, y) \land \neg At(b, x) \land \neg At(Shakey, x))\\
&Action(\text{ACTION:}ClimbUp(b), \text{PRECOND:}At(Shakey, x) \land At(b, x) \land Climbable(b),\\
&\quad \text{EFFECT:}On(Shakey, b) \land \neg On(Shakey, Floor))\\
&Action(\text{ACTION:}ClimbDown(b), \text{PRECOND:}On(Shakey, b),\\
&\quad \text{EFFECT:}On(Shakey, Floor) \land \neg On(Shakey, b))\\
&Action(\text{ACTION:}TurnOn(l), \text{PRECOND:}On(Shakey, b) \land At(Shakey, x) \land At(l, x),\\
&\quad \text{EFFECT:}TurnedOn(l))\\
&Action(\text{ACTION:}TurnOff(l), \text{PRECOND:}On(Shakey, b) \land At(Shakey, x) \land At(l, x),\\
&\quad \text{EFFECT:}\neg TurnedOn(l))
\end{aligned}$$

The initial state is:

$$
\begin{array}{l}
In(Switch_1, Room_1) \wedge In(Door_1, Room_1) \wedge In(Door_1, Corridor)\\
In(Switch_1, Room_2) \wedge In(Door_2, Room_2) \wedge In(Door_2, Corridor)\\
In(Switch_1, Room_3) \wedge In(Door_3, Room_3) \wedge In(Door_3, Corridor)\\
In(Switch_1, Room_4) \wedge In(Door_4, Room_4) \wedge In(Door_4, Corridor)\\
In(Shakey, Room_3) \wedge At(Shakey, X_S)\\
In(Box_1, Room_1) \wedge In(Box_2, Room_1) \wedge In(Box_3, Room_1) \wedge In(Box_4, Room_1)\\
Climbable(Box_1) \wedge Climbable(Box_2) \wedge Climbable(Box_3) \wedge Climbable(Box_4)\\
Pushable(Box_1) \wedge Pushable(Box_2) \wedge Pushable(Box_3) \wedge Pushable(Box_4)\\
At(Box_1, X_1) \wedge At(Box_2, X_2) \wedge At(Box_3, X_3) \wedge At(Box_4, X_4)\\
TurnwdOn(Switch_1) \wedge TurnedOn(Switch_4)
\end{array}
$$

A plan to achieve the goal is:

$$
\begin{array}{l}
Go(X_S, Door_3)\\
Go(Door_3, Door_1)\\
Go(Door_1, X_2)\\
Push(Box_2, X_2, Door_1)\\
Push(Box_2, Door_1, Door_2)\\
Push(Box_2, Door_2, Switch_2)
\end{array}
$$

Exercise 11.FTUM

A finite Turing machine has a finite one-dimensional tape of cells, each cell containing one of a finite number of symbols. One cell has a read and write head above it. There is a finite set of states the machine can be in, one of which is the accept state. At each time step, depending on the symbol on the cell under the head and the machine's current state, there are a set of actions we can choose from. Each action involves writing a symbol to the cell under the head, transitioning the machine to a state, and optionally moving the head left or right. The mapping that determines which actions are allowed is the Turing machine's program. Your goal is to control the machine into the accept state.

Represent the Turing machine acceptance problem as a planning problem. If you can do this, it demonstrates that determining whether a planning problem has a solution is at least as hard as the Turing acceptance problem, which is PSPACE-hard.

One representation is as follows. We have the predicates:

a. $HeadAt(c)$: tape head at cell location c, true for exactly one cell.

b. $State(s)$: machine state is s, true for exactly one cell.

c. $ValueOf(c, v)$: cell c's value is v.

d. $LeftOf(c_1, c_2)$: cell c_1 is one step left from cell c_2.

e. $TransitionLeft(s_1, v_1, s_2, v_2)$: the machine in state s_1 upon reading a cell with value v_1 may write value v_2 to the cell, change state to s_2, and transition to the left.

f. $\mathit{TransitionRight}(s_1, v_1, s_2, v_2)$: the machine in state s_1 upon reading a cell with value v_1 may write value v_2 to the cell, change state to s_2, and transition to the right.

The predicates HeadAt, State, and $\mathit{ValueOf}$ are fluents, the rest are constant descriptions of the machine and its tape. Two actions are required:

$$\begin{aligned}
&\mathit{Action}(\mathit{RunLeft}(s_1, c_1, v_1, , s_2, c_2, v_2),\\
&\quad \text{PRECOND:}\mathit{State}(s_1) \land \mathit{HeadAt}(c_1) \land \mathit{ValueOf}(c_1, v_1)\\
&\qquad\quad ;\ \land \mathit{TransitionLeft}(s_1, v_1, s_2, v_2) \land \mathit{LeftOf}(c_2, c_1)\\
&\quad \text{EFFECT:}\neg \mathit{State}(s_1) \land \mathit{State}(s_2) \land \neg \mathit{HeadAt}(c_1) \land \mathit{HeadAt}(c_2)\\
&\qquad\quad \land\neg\, \mathit{ValueOf}(c_1, v_1) \land \mathit{ValueOf}(c_1, v_2))
\end{aligned}$$

$$\begin{aligned}
&\mathit{Action}(\mathit{RunRight}(s_1, c_1, v_1, , s_2, c_2, v_2),\\
&\quad \text{PRECOND:}\mathit{State}(s_1) \land \mathit{HeadAt}(c_1) \land \mathit{ValueOf}(c_1, v_1)\\
&\qquad\quad ;\ \land \mathit{TransitionRight}(s_1, v_1, s_2, v_2) \land \mathit{LeftOf}(c_1, c_2)\\
&\quad \text{EFFECT:}\neg \mathit{State}(s_1) \land \mathit{State}(s_2) \land \neg \mathit{HeadAt}(c_1) \land \mathit{HeadAt}(c_2)\\
&\qquad\quad \land\neg\, \mathit{ValueOf}(c_1, v_1) \land \mathit{ValueOf}(c_1, v_2))
\end{aligned}$$

The goal will typically be to reach a fixed accept state. A simple example problem is:

$$\begin{aligned}
&\mathit{Init}(\mathit{HeadAt}(C_0) \land \mathit{State}(S_1) \land \mathit{ValueOf}(C_0, 1) \land \mathit{ValueOf}(C_1, 1)\\
&\quad \land \mathit{ValueOf}(C_2, 1) \land \mathit{ValueOf}(C_3, 0) \land \mathit{LeftOf}(C_0, C_1) \land \mathit{LeftOf}(C_1, C_2)\\
&\quad \land \mathit{LeftOf}(C_2, C_3) \land \mathit{TransitionLeft}(S_1, 1, S_1, 0) \land \mathit{TransitionLeft}(S_1, 0, S_{\text{accept}}, 0)\\
&\mathit{Goal}(\mathit{State}(S_{\text{accept}}))
\end{aligned}$$

Note that the number of literals in a state is linear in the number of cells, which means a polynomial space machine require polynomial state to represent.

Exercise 11.HOLD

The goals we have considered so far all ask the planner to make the world satisfy the goal at just one time step. Not all goals can be expressed this way: you do not achieve the goal of suspending a chandelier above the ground by throwing it in the air. More seriously, you wouldn't want your spacecraft life-support system to supply oxygen one day but not the next. A *maintenance goal* is achieved when the agent's plan causes a condition to hold continuously from a given state onward. Describe how to extend the formalism of this chapter to support maintenance goals.

The simplest extension allows for maintenance goals that hold in the initial state and must remain true throughout the execution of the plan. Safety goals (do no harm) are typically of this form. This extends classical planning problems to allow a maintenance goal. A plan solves the problem if the final state satisfies the regular goals, and all visited states satisfy the maintenance goal.

The life-support example cannot, however, be solved by a finite plan. An extension to infinite plans can capture this, where an infinite plan solves a planning problem if the goal is eventually satisfied by the plan, i.e., there is a point after which the goal is continuously true. Infinite solutions can be described finitely with loops.

For the chandelier example we can allow NoOp actions which do nothing except model the passing of physics. The idea is that a solution will have a finite prefix with an infinite tail (i.e., a loop) of NoOps. This will allow the problem specification to capture the instability of a thrown chandelier, as after a certain number of time steps it would no longer be suspended.

Exercise 11.PLOP

Some of the operations in standard programming languages can be modeled as actions that change the state of the world. For example, the assignment operation changes the contents of a memory location, and the print operation changes the state of the output stream. A program consisting of these operations can also be considered as a plan, whose goal is given by the specification of the program. Therefore, planning algorithms can be used to construct programs that achieve a given specification.

a. Write an action schema for the assignment operator (assigning the value of one variable to another). Remember that the original value will be overwritten!

b. Show how object creation can be used by a planner to produce a plan for exchanging the values of two variables by using a temporary variable.

We need one action, $Assign$, which assigns the value in the source register (or variable if you prefer, but the term "register" makes it clearer that we are dealing with a physical location) sr to the destination register dr:

$$\begin{array}{l} Action(\text{ACTION:}Assign(dr, sr), \\ \quad \text{PRECOND:}Register(dr) \land Register(sr) \land Value(dr, dv) \land Value(sr, sv), \\ \quad \text{EFFECT:}Value(dr, sv) \land \neg Value(dr, dv)) \end{array}$$

Now suppose we start in an initial state with $Register(R_1) \land Register(R_2) \land Value(R_1, V_1) \land Value(R_2, V_2)$ and we have the goal $Value(R_1, V_2) \land Value(R_2, V_1)$. Unfortunately, there is no way to solve this as is. We either need to add an explicit $Register(R_3)$ condition to the initial state, or we need a way to create new registers. That could be done with an action for allocating a new register:

$$\begin{array}{l} Action(\text{ACTION:}Allocate(r), \\ \quad \text{EFFECT:}Register(r)) \end{array}$$

Then the following sequence of steps constitues a valid plan:

$$\begin{array}{l} Allocate(R_3) \\ Assign(R_3, R_1) \\ Assign(R_1, R_2) \\ Assign(R_2, R_1) \end{array}$$

11.2 Algorithms for Classical Planning

Exercise 11.SUSS

Figure 11.3 (page 365) shows a blocks-world problem that is known as the **Sussman anomaly**. The problem was considered anomalous because the noninterleaved planners of the early 1970s could not solve it. Write a definition of the problem and solve it, either by hand or with a planning program. A noninterleaved planner is a planner that, when given two subgoals G_1 and G_2, produces either a plan for G_1 concatenated with a plan for G_2, or vice versa. Explain why a noninterleaved planner cannot solve this problem.

The initial state is:

$$On(B, Table) \land On(C, A) \land On(A, Table) \land Clear(B) \land Clear(C)$$

The goal is:

$$On(A, B) \land On(B, C)$$

First we'll explain why it is an anomaly for a noninterleaved planner. There are two subgoals; suppose we decide to work on $On(A, B)$ first. We can clear C off of A and then move A on to B. But then there is no way to achieve $On(B, C)$ without undoing the work we have done. Similarly, if we work on the subgoal $On(B, C)$ first we can immediately achieve it in one step, but then we have to undo it to get A on B.

Now we'll show how things work out with an interleaved planner such as POP. Since $On(A, B)$ isn't true in the initial state, there is only one way to achieve it: $Move(A, x, B)$, for some x. Similarly, we also need a $Move(B, x', C)$ step, for some x'. Now let's look at the $Move(A, x, B)$ step. We need to achieve its precondition $Clear(A)$. We could do that either with $Move(b, A, y)$ or with $MoveToTable(b, A)$. Let's assume we choose the latter. Now if we bind b to C, then all of the preconditions for the step $MoveToTable(C, A)$ are true in the initial state, and we can add causal links to them. We then notice that there is a threat: the $Move(B, x', C)$ step threatens the $Clear(C)$ condition that is required by the $MoveToTable$ step. We can resolve the threat by ordering $Move(B, x', C)$ after the $MoveToTable$ step. Finally, notice that all the preconditions for $Move(B, x', C)$ are true in the initial state. Thus, we have a complete plan with all the preconditions satisfied. It turns out there is a well-ordering of the three steps:

$$\begin{array}{l} MoveToTable(C, A) \\ Move(B, Table, C) \\ Move(A, Table, B) \end{array}$$

Exercise 11.BSPD

Prove that backward search with PDDL problems is complete.

Briefly, the reason is the same as for forward search: in the absence of function symbols, a PDDL state space is finite. Hence any complete search algorithm will be complete for PDDL planning, whether forward or backward.

Exercise 11.BIDP

Examine the definition of **bidirectional search** in Chapter 3.

a. Would bidirectional state-space search be a good idea for planning?

b. What about bidirectional search in the space of partial-order plans?

c. Devise a version of partial-order planning in which an action can be added to a plan if its preconditions can be achieved by the effects of actions already in the plan. Explain how to deal with conflicts and ordering constraints. Is the algorithm essentially identical to forward state-space search?

a. It is feasible to use bidirectional search, because it is possible to invert the actions. However, most of those who have tried have concluded that biderectional search is generally not efficient, because the forward and backward searches tend to miss each other. This is due to the large state space. A few planners, such as PRODIGY (Fink and Blythe, 1998) have used bidirectional search.

b. Again, this is feasible but not popular. PRODIGY is in fact (in part) a partial-order planner: in the forward direction it keeps a total-order plan (equivalent to a state-based planner), and in the backward direction it maintains a tree-structured partial-order plan.

c. An action A can be added if all the preconditions of A have been achieved by other steps in the plan. When A is added, ordering constraints and causal links are also added to make sure that A appears after all the actions that enabled it and that a precondition is not disestablished before A can be executed. The algorithm does search forward, but it is not the same as forward state-space search because it can explore actions in parallel when they don't conflict. For example, if A has three preconditions that can be satisfied by the non-conflicting actions B, C, and D, then the solution plan can be represented as a single partial-order plan, while a state-space planner would have to consider all 3! permutations of B, C, and D.

Exercise 11.FBSS

We contrasted forward and backward state-space searchers with partial-order planners, saying that the latter is a plan-space searcher. Explain how forward and backward state-space search can also be considered plan-space searchers, and say what the plan refinement operators are.

A forward state-space planner maintains a partial plan that is a strict linear sequence of actions; the plan refinement operator is to add an applicable action to the end of the sequence, updating literals according to the action's effects.

A backward state-space planner maintains a partial plan that is a reversed sequence of actions; the refinement operator is to add an action to the beginning of the sequence as long as the action's effects are compatible with the state at the beginning of the sequence.

Exercise 11.SATX

Up to now we have assumed that the plans we create always make sure that an action's preconditions are satisfied. Let us now investigate what propositional successor-state axioms such as $HaveArrow^{t+1} \Leftrightarrow (HaveArrow^{t} \wedge \neg Shoot^{t})$ have to say about actions whose preconditions are not satisfied.

a. Show that the axioms predict that nothing will happen when an action is executed in a state where its preconditions are not satisfied.

b. Consider a plan p that contains the actions required to achieve a goal but also includes illegal actions. Is it the case that

$$\textit{initial state} \wedge \textit{successor-state axioms} \wedge p \models \textit{goal}\ ?$$

c. With first-order successor-state axioms in situation calculus, is it possible to prove that a plan containing illegal actions will achieve the goal?

a. We can illustrate the basic idea using the axiom given. Suppose that $Shoot^{t}$ is true but $HaveArrow^{t}$ is false. Then the RHS of the axiom is false, so $HaveArrow^{t+1}$ is false, as we would hope. More generally, if an action precondition is violated, then both $ActionCausesF^{t}$ and $ActionCausesNotF^{t}$ are false, so the generic successor-state axiom reduces to

$$F^{t+1} \Leftrightarrow False \vee (F^{t} \wedge True)\ .$$

which is the same as saying $F^{t+1} \Leftrightarrow F^{t}$, i.e., nothing happens.

b. Yes, the plan plus the axioms will entail goal satisfaction; the axioms will copy every fluent across an illegal action and the rest of the plan will still work. Note that goal entailment is trivially achieved if we add precondition axioms, because then the plan is logically inconsistent with the axioms and every sentence is entailed by a contradiction. Precondition axioms are a way to *prevent* illegal actions in satisfiability-based planning methods.

c. No. As written in Section 10.4.2, the sucessor-state axioms preclude proving anything about the outcome of a plan with illegal actions. When $Poss(a, s)$ is false, the axioms say nothing about the situation resulting from the action.

Exercise 11.STTR

Consider how to translate a set of action schemas into the successor-state axioms of situation calculus.

a. Consider the schema for $Fly(p, from, to)$. Write a logical definition for the predicate $Poss(Fly(p, from, to), s)$, which is true if the preconditions for $Fly(p, from, to)$ are satisfied in situation s.

b. Next, assuming that $Fly(p, from, to)$ is the only action schema available to the agent, write down a successor-state axiom for $At(p, x, s)$ that captures the same information as the action schema.

c. Now suppose there is an additional method of travel: $Teleport(p, from, to)$. It has the additional precondition $\neg Warped(p)$ and the additional effect $Warped(p)$. Explain how the situation calculus knowledge base must be modified.

d. Finally, develop a general and precisely specified procedure for carrying out the translation from a set of action schemas to a set of successor-state axioms.

The main point here is that writing each successor-state axiom correctly requires knowing *all* the actions that might add or delete a given fluent; writing a STRIPS axiom, on the other hand, requires knowing *all* the fluents that a given action might add or delete.

a.

$$\begin{aligned}&Poss(Fly(p, from, to), s) \;\Leftrightarrow\\ &\quad At(p, from, s) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)\ .\end{aligned}$$

b.

$$\begin{aligned}&Poss(a, s) \;\Rightarrow\\ &\quad (At(p, to, Result(a, s)) \Leftrightarrow\\ &\qquad (\exists\, from \;\; a = Fly(p, from, to)) \vee\\ &\qquad (At(p, to, s) \wedge \neg\exists\, new \;\; new \neq to \wedge a = Fly(p, to, new)))\ .\end{aligned}$$

c. We must add the possibility axiom for the new action:

$$\begin{aligned}&Poss(Teleport(p, from, to), s) \;\Leftrightarrow\\ &\quad At(p, from, s) \wedge \neg Warped(p, s) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)\ .\end{aligned}$$

The successor-state axiom for location must be revised:

$$\begin{aligned}&Poss(a, s) \;\Rightarrow\\ &\quad (At(p, to, Result(a, s)) \Leftrightarrow\\ &\qquad (\exists\, from \;\; a = Fly(p, from, to)) \vee\\ &\qquad (\exists\, from \;\; a = Teleport(p, from, to)) \vee\\ &\qquad (At(p, to, s) \wedge \neg\exists\, new \;\; new \neq to \wedge\\ &\qquad\quad (a = Fly(p, to, new) \vee a = Teleport(p, to, new))))\ .\end{aligned}$$

Finally, we must add a successor-state axiom for Warped:

$$\begin{aligned}&\mathit{Poss}(a,s) \Rightarrow\\ &\quad (\mathit{Warped}(p,\mathit{Result}(a,s)) \Leftrightarrow\\ &\qquad (\exists\, \mathit{from}, \mathit{to}\;\; a = \mathit{Teleport}(p,\mathit{from},\mathit{to})) \lor \mathit{Warped}(p,s))\ .\end{aligned}$$

d. The basic procedure is essentially given in the description of classical planning as Boolean satisfiability in 10.4.1, except that there is no grounding step, the precondition axioms become definitions of Poss for each action, and the successor-state axioms use the structure given in 10.4.2 with existential quantifiers for all free variables in the actions, as shown in the examples above.

Exercise 11.SATD

In the SATPLAN algorithm in Figure 7.22 (page 262), each call to the satisfiability algorithm asserts a goal g^T, where T ranges from 0 to $T_{\max}$. Suppose instead that the satisfiability algorithm is called only once, with the goal $g^0 \lor g^1 \lor \cdots \lor g^{T_{\max}}$.

a. Will this always return a plan if one exists with length less than or equal to $T_{\max}$?

b. Does this approach introduce any new spurious "solutions"?

c. Discuss how one might modify a satisfiability algorithm such as WALKSAT so that it finds short solutions (if they exist) when given a disjunctive goal of this form.

a. Yes, this will find a plan whenever the normal SATPLAN finds a plan no longer than T_{max}.

b. This will not cause SATPLAN to return an incorrect solution, but it might lead to plans that, for example, achieve and unachieve the goal several times.

c. There is no simple and clear way to induce WALKSAT to find short solutions, because it has no notion of the length of a plan—the fact that the problem is a planning problem is part of the encoding, not part of WALKSAT. But if we are willing to do some rather brutal surgery on WALKSAT, we can achieve shorter solutions by identifying the variables that represent actions and (1) tending to randomly initialize the action variables (particularly the later ones) to false, and (1) preferring to randomly flip an earlier action variable rather than a later one.

11.3 Heuristics for Planning

Exercise 11.NEGE

Explain why dropping negative effects from every action schema in a planning problem results in a relaxed problem.

Goals and preconditions can only be positive literals. So a negative effect can only make it harder to achieve a goal (or a precondition to an action that achieves the goal). Therefore, eliminating all negative effects only makes a problem easier. This would *not* be true if negative preconditions and goals were allowed.

11.4 Hierarchical Planning

Exercise 11.FDEX

You have a number of trucks with which to deliver a set of packages. Each package starts at some location on a grid map, and has a destination somewhere else. Each truck is directly controlled by moving forward and turning. Construct a hierarchy of high-level actions for this problem. What knowledge about the solution does your hierarchy encode?

We first need to specify the primitive actions: for movement we have $\mathit{Forward}(t)$, $\mathit{TurnLeft}(t)$, and $\mathit{TurnRight}(t)$ where t is a truck, and for package delivery we have $\mathit{Load}(p,t)$ and $\mathit{Unload}(p,t)$ where p is a package and t is a truck. These can be given PDDL descriptions in the usual way.

The hierarchy can be built in a number of ways, but one is to use the HLA $\mathit{Navigate}(t,[x,y])$ to take a truck t to coordinates $[x,y]$, and $\mathit{Deliver}(t,p)$ to deliver package p to its destination with truck t. We assume the fluent $\mathit{At}(o,[x,y])$ for trucks and packages o records their current position $[x,y]$, the predicate $\mathit{Destination}(p,[x',y'])$ gives the package's destination.

This hierarchy (Figure S**??**) encodes the knowledge that trucks can only carry one package at a time, that we need only drop packages off at their destinations not intermediate points, and that we can serialize deliveries (in reality, trucks would move in parallel, but we have no representation for parallel actions here). From a higher-level, the hierarchy says that the planner needs only to choose which trucks deliver which packages in what order, and trucks should navigate given their destinations.

Exercise 11.HLAU

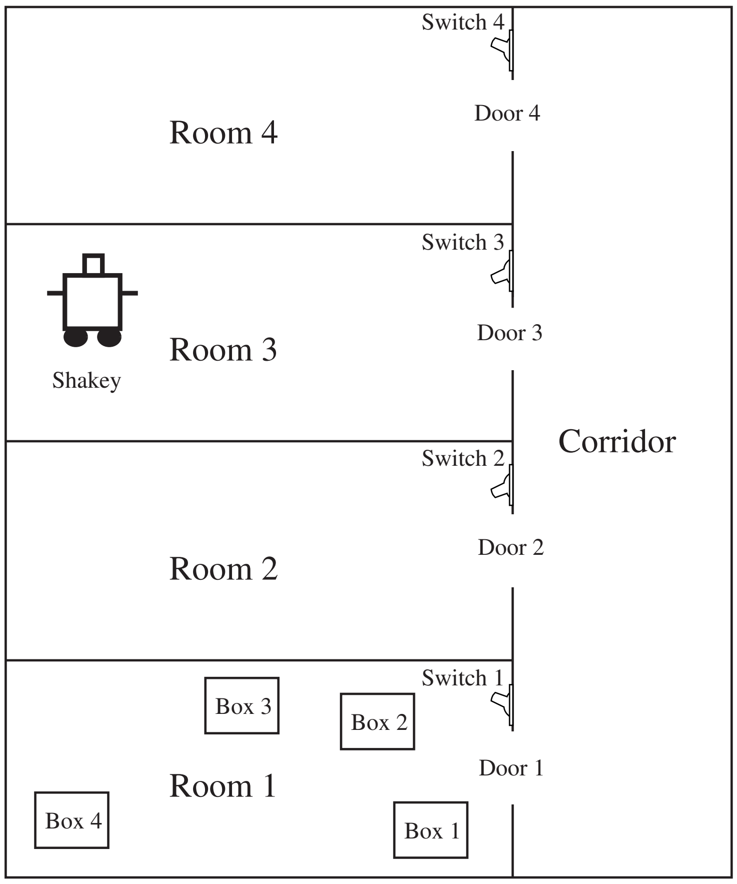
Suppose that a high-level action has exactly one implementation as a sequence of primitive actions. Give an algorithm for computing its preconditions and effects, given the complete refinement hierarchy and schemas for the primitive actions.

To simplify the problem, we assume that at most one refinement of a high-level action will be applicable at a given time (not much of a restriction since there is a unique solution).

The algorithm shown below maintains at each point the net preconditions and effects of the prefix of h processed so far. This includes both preconditions and effects of primitive actions, and preconditions of refinements. Note that any literal not in effect is untouched by the prefix currently processed.

```
net_preconditions <- {}
net_effects <- {}
remaining <- [h]

while remaining not empty:
  a <- pop remaining
```



Shakey's world. Shakey can move between landmarks within a room, can pass through the door between rooms, can climb climbable objects and push pushable objects, and can flip light switches.

```
if a is primitive:
  add to net_preconditions any precondition of a not in effects
  add to net_effects the effects of action a, first removing any
       complementary literals
else:
  r <- the unique refinement whose preconditions do not include
       literals negated in net_effect or net_preconditions
  add to net_preconditions any preconditions of r not in effect
  prepend to remaining the sequence of actions in r
```

Exercise 11.OPTR

Suppose that the optimistic reachable set of a high-level plan is a superset of the goal set; can anything be concluded about whether the plan achieves the goal? What if the pessimistic reachable set doesn't intersect the goal set? Explain.

We cannot draw any conclusions. Just knowing that the optimistic reachable set is a superset of the goal is no more help than knowing only that it intersects the goal: the optimistic reachable set only guarantees that we cannot reach states outside of it, not that we can reach any of the states inside it. Similarly, the pessimistic reachable set only says we can definitely reach state inside of it, not that we cannot reach states outside of it.

Exercise 11.HLAP

Write an algorithm that takes an initial state (specified by a set of propositional literals) and a sequence of HLAs (each defined by preconditions and angelic specifications of optimistic and pessimistic reachable sets) and computes optimistic and pessimistic descriptions of the reachable set of the sequence.

To simplify, we don't model HLA precondition tests. (Comparing the preconditions to the optimistic and pessimistic descriptions can sometimes determine if preconditions are definitely or definitely not satisfied, respectively, but may be inconclusive.)

The operation to propagate 1-CNF descriptions through descriptions is the same for optimistic and pessimistic descriptions, and is as follows:

```
state <- initial state

for each HLA h in order:
  for each literal in the description of h:
    choose case depending on form of literal:
      +l:              state <- state - {-l} + {l}
      -l:              state <- state - {l} + {-l}
      poss add l:      state <- state + {l}
      poss del l:      state <- state + {-l}
      poss add del l: state <- state + {l,-l}

description <- conjunction of all literals which are
               not part of a complementary pair in state
```

11.5 Planning and Acting in Nondeterministic Domains

Exercise 11.NDTE

Consider the following argument: In a framework that allows uncertain initial states, **nondeterministic effects** are just a notational convenience, not a source of additional representational power. For any action schema a with nondeterministic effect $P \lor Q$, we could always replace it with the conditional effects **when** R: $P \land$ **when** $\neg R$: Q, which in turn can be reduced to two regular actions. The proposition R stands for a random proposition that is unknown in the initial state and for which there are no sensing actions. Is this argument correct? Consider separately two cases, one in which only one instance of action schema a is in the plan, the other in which more than one instance is.

It is equivalent in the first case, by the argument given above. However, if there are two or more copies of the same schema in a plan it is different: all actions will have the same nondeterministic effect: if R is true both will result in P, otherwise both result in Q. To allow copies of the same schema to differ we need one random variable per copy.

Exercise 11.FLIP

Suppose the *Flip* action always changes the truth value of variable L. Show how to define its effects by using an action schema with conditional effects. Show that, despite the use of conditional effects, a 1-CNF belief state representation remains in 1-CNF after a *Flip*.

Flip can be described using conditional effects:

$Action(Flip,$
  EFFECT:**when** L: $\neg L \wedge$ **when** $\neg L$: $L)$.

To see that a 1-CNF belief state representation stays 1-CNF after *Flip*, observe that there are three cases. If L is true in the belief state, then it is false after *Flip*; conversely if it is false. Finally, if L is unknown before, then it is unknown after: either L or $\neg L$ can obtain. All other components of the belief state remain unchanged, since it is 1-CNF.

Exercise 11.BLKA

In the blocks world we were forced to introduce two action schemas, *Move* and *MoveToTable*, in order to maintain the *Clear* predicate properly. Show how conditional effects can be used to represent both of these cases with a single action.

Using the second definition of *Clear* in the chapter—namely, that there is a clear space for a block—the only change is that the destination remains clear if it is the table:

$Action(Move(b, x, y),$
  PRECOND:$On(b, x) \wedge Clear(b) \wedge Clear(y),$
  EFFECT:$On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge ($**when** $y \neq Table$: $\neg Clear(y)))$

Exercise 11.ALTV

Conditional effects were illustrated for the *Suck* action in the vacuum world—which square becomes clean depends on which square the robot is in. Can you think of a new set of propositional variables to define states of the vacuum world, such that *Suck* has an *unconditional* description? Write out the descriptions of *Suck*, *Left*, and *Right*, using your propositions, and demonstrate that they suffice to describe all possible states of the world.

Let $CleanH$ be true iff the robot's current square is clean and $CleanO$ be true iff the other square is clean. Then *Suck* is characterized by

$Action(Suck,$ PRECOND:, EFFECT:$CleanH)$

Unfortunately, moving affects these new literals! For $Left$ we have

$$\begin{aligned}&Action(Left, \textsc{Precond}{:}AtR,\\ &\quad \textsc{Effect}{:}AtL \wedge \neg AtR \wedge \textbf{when } CleanH{:}\ CleanO \wedge \textbf{when } CleanO{:}\ CleanH\\ &\qquad \wedge\ \textbf{when } \neg CleanO{:}\ \neg CleanH \wedge \textbf{when } \neg CleanH{:}\ \neg CleanO)\end{aligned}$$

with the dual for $Right$.

Exercise 11.DIRT

Find a suitably dirty carpet, free of obstacles, and vacuum it. Draw the path taken by the vacuum cleaner as accurately as you can. Explain it, with reference to the forms of planning discussed in this chapter.

The main thing to notice here is that the vacuum cleaner moves repeatedly over dirty areas—presumably, until they are clean. Also, each forward move is typically short, followed by an immediate reversing over the same area. This is explained in terms of a disjunctive outcome: the area may be fully cleaned or not, the reversing enables the agent to check, and the repetition ensures completion (unless the dirt is ingrained). Thus, we have a strong cyclic plan with sensing actions.

Exercise 11.SHAM

The following quotes are from the backs of shampoo bottles. Identify each as an unconditional, conditional, or execution-monitoring plan. (a) "Lather. Rinse. Repeat." (b) "Apply shampoo to scalp and let it remain for several minutes. Rinse and repeat if necessary." (c) "See a doctor if problems persist."

(a) Literally its an unconditional plan, but generally people would interpret it similarly to (b).

(b) Conditional: the final action loops to the start only if "necessary".

(c) This can be seen as a conditional plan. But it can also be seen as execution monitoring: we continue treatment until the problem resolves itself. If we notice the problem persisting, we complete the plan.

Exercise 11.DOCO

Consider the following problem: A patient arrives at the doctor's office with symptoms that could have been caused either by dehydration or by disease D (but not both). There are two possible actions: $Drink$, which unconditionally cures dehydration, and $Medicate$, which cures disease D but has an undesirable side effect if taken when the patient is dehydrated. Write the problem description, and diagram a sensorless plan that solves the problem, enumerating all relevant possible worlds.

The two actions can be represented as:

$$\begin{array}{l} Action(Drink, \\ \qquad \text{EFFECT:}\textbf{when } \neg D\text{: } Cured)\ . \\ Action(Medicate, \\ \qquad \text{EFFECT:}\textbf{when } D\text{: } Cured \land \textbf{when } D\text{: } SideEffect)\ . \end{array}$$

The goal is $Cured \land \neg SideEffect$.

The plan $[Drink, Medicate]$ solves the problem. To see this, note that the relevant possible worlds before executing the plan are $\{D, \neg D\}$. After executing *Drink* the worlds are $\{D, \neg D \land Cured\}$ and after *Medicate* they are $\{D \land Cured, \neg D \land Cured\}$.

Exercise 11.MEDI

To the medication problem in the previous exercise, add a *Test* action that has the conditional effect *CultureGrowth* when *Disease* is true and in any case has the perceptual effect $Known(CultureGrowth)$. Diagram a conditional plan that solves the problem and minimizes the use of the *Medicate* action.

One solution plan is $[Test, \textbf{if} CultureGrowth \textbf{then} [Drink, Medicate]]$.

11.6 Time, Schedules, and Resources

Exercise 11.CPMJ

In Figure 11.14 we showed how to describe actions in a scheduling problem by using separate fields for DURATION, USE, and CONSUME. Now suppose we wanted to combine scheduling with nondeterministic planning, which requires nondeterministic and conditional effects. Consider each of the three fields and explain if they should remain separate fields, or if they should become effects of the action. Give an example for each of the three.

The natural nondeterministic generalization of DURATION, USE, and CONSUME represents each as an *interval* of possible values rather than a single value. Algorithms that work with quantities can all be modified relatively easily to manage intervals over quantities—for example, by representing them as inequalities for the lower and upper bounds. Thus, if the agent starts with 10 screws and the first action in a plan consumes 2–4 screws, then a second action requiring 5 screws is still executable.

When it comes to conditional effects, however, the fields must be treated differently. The USE field refers to a constraint holding *during* the action, rather than *after* it is done. Thus, it has to remain a separate field, since it is not treated in the same way as an effect. The DURATION and CONSUME fields both describe effects (on the clock and on the quantity of a resource); thus, they can be folded into the conditional effect description for the action.

[[need exercises]]

11.7 Analysis of Planning Approaches

[[need exercises]]