Planning (Ch. 10)
Planning

Planning is doing a sequence of actions to achieve one or more goals

This differs from search in that there are often multiple objectives that must be done (states can have similarities, not just “different”)

You can always reduce a planning problem to a search problem, but this is quite often very expensive
Search

Search: How to get from point A to point B quickly? (Only considering traveling)
Planning

Planning: multiple tasks/subtasks need to be done and in what order? (pack, travel, unpack)
Search vs planning

Searching: finding a single goal
Planning: must complete multiple tasks on the way to an ultimate goal
Planning: definitions

The book uses Planning Domain Definition Language (PDDL) to represent states/actions.

PDDL is very similar to first order logic in terms of notation (states are now similar to what our knowledge base was).

The large difference is that we need to define actions to move between states.
Planning: state

A state is all of the facts ANDed together in FO logic, but want to avoid:
1. Variables (otherwise it would not be specific)
2. Functions (just replace them with objects)
3. Negations (as we assume everything not mentioned is false)

State = $BK_{\text{Knight}}(D, 8) \land BP_{\text{Pawn}}(C, 7) \land BK_{\text{ing}}(D, 7) \land WP_{\text{awn}}(B, 6) \land W_{\text{Knight}}(C, 6) \land W_{\text{Rook}}(E, 6) \land \ldots \land Turn(Black)$
Planning: actions

Actions have three parts:
1. Name (similar to a function call)
2. Precondition (requirements to use action)
3. Effect (unmentioned states do not change)

For example:

Action( MoveBK\text{night}RRD(x, y),

Precondition: \text{BKnight}(x, y) \land \text{Turn(Black)},

Effect: \neg\text{BKnight}(x, y) \land \text{BKnight}(x + 2, y - 1) \\
\land \neg\text{Turn(Black)} \land \text{Turn(White)}

\rightarrow remove black's turn
Planning: actions

State = \( B\text{Knight}(D, 8) \land B\text{Pawn}(C, 7) \land B\text{King}(D, 7) \land W\text{Pawn}(B, 6) \land W\text{Knight}(C, 6) \land W\text{Rook}(E, 6) \land \ldots \land \text{Turn(Black)} \)

Apply: \( \text{Move} B\text{Knight} \text{RRD}(D, 8) \)

State = \( B\text{Knight}(F, 7) \land B\text{Pawn}(C, 7) \land B\text{King}(D, 7) \land W\text{Pawn}(B, 6) \land W\text{Knight}(C, 6) \land W\text{Rook}(E, 6) \land \ldots \land \text{Turn(White)} \)
Planning: example

Let's look at a grocery store example:

Objects = store locations and food items

Goal = $\text{At(Checkout)} \land \text{Cart(Milk)} \land \text{Cart(Apples)}$

$\land \text{Cart(Eggs)} \land \text{Cart(ToiletPaper)} \land \text{Cart(Bananas)}$

$\land \text{Cart(Bread)} \land \neg \text{Cart(Candy)}$

Aisle 1 = Milk, Eggs
Aisle 2 = Apples, Bananas
Aisle 3 = Bread, Candy, ToiletPaper
<table>
<thead>
<tr>
<th>Action</th>
<th>Precondition</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>GoTo(x, y)</td>
<td>At(x)</td>
<td>$\neg At(x) \land At(y)$</td>
</tr>
<tr>
<td>AddApples()</td>
<td>At(Aisle2)</td>
<td>Cart(Apples)</td>
</tr>
<tr>
<td>AddMilk()</td>
<td>At(Aisle1)</td>
<td>Cart(Milk)</td>
</tr>
<tr>
<td>AddBananas()</td>
<td>At(Aisle2)</td>
<td>Cart(Bananas)</td>
</tr>
<tr>
<td>AddEggs()</td>
<td>At(Aisle1)</td>
<td>Cart(Eggs)</td>
</tr>
<tr>
<td>AddBread()</td>
<td>At(Aisle3)</td>
<td>Cart(Bread)</td>
</tr>
<tr>
<td>AddCandy()</td>
<td>At(Aisle3)</td>
<td>Cart(Candy)</td>
</tr>
<tr>
<td>AddToiletPaper()</td>
<td>At(Aisle3)</td>
<td>Cart(ToiletPaper)</td>
</tr>
</tbody>
</table>
Planning: example

Initial state = At(Door)
A possible solution:
1. GoTo(Aisle1)  
2. Add(Milk)  
3. Add(Eggs)  
4. GoTo(Aisle2)  
5. Add(Apples)  
6. GoTo(Aisle3)  
7. Add(Bread)  
8. Add(ToiletPaper)  
9. GoTo(Aisle2)  
10. Add(Bananas)  
11. GoTo(Checkout)

Not most efficient, but goal reached
Planning: decidability

Since our planning is similar to FO logic, it is unsurprisingly semi-decidable as well.

Thus, in general you will be able to find a solution if it exists, but possibly be unable to tell if a solution does not exist.

If there are no functions or we know the goal can be found in a finite number of steps, then it is decidable.
Planning: actions

If we treat the current state like a knowledge base and actions with $\forall$s for every variable...

“state entails $\text{Precondition}(A)$” means action A's preconditions are met for the state.

Thus if each action uses $v$ variables, each with $k$ possible values, there are $O(k^v)$ actions (we can ignore actions that do not change the current state in some cases).
Planning: difficulty

PlanSAT tells whether a solution exists or not, but takes PSPACE to tell

If negative preconditions are not allowed, we find a solution in P, and optimal in NP-hard
Planning: algorithms

Again similar to FO logic, there are two basic algorithms you can use to try and plan:

1. Forward search - similar to BFS and check all states you can find in 1 action, then 2 actions, then 3... until you find the goal state

2. Backward search - start at goal and try to work backwards to initial state
Forward search

Forward search is a brute force search that finds all possible states you can end up in.

Each action is tested on each state currently known and is repeated until the goal is found.

This can be quite costly, as actions that do not lead to the goal could be repeatedly explored. (We will see a way to improve this.)
Forward search

At(Door) — GoTo(Door) — can ignore

At(Door) — GoTo(Checkout)

At(Aisle1) — AddMilk()
Cart(Milk)

At(Aisle2) — ...

At(Aisle3) — ...

At(Checkout) — ...

At(Aisle1) — ...
Forward search

You try it!

Initial: At(Truck, UPSD) ^ Package(UPSD, P1) ^ Package(UPSD, P2) ^ Mobile(Truck)

Goal: Package(H1, P1) ^ Package(H2, P2)

Action( GoTo(x, y, z),
Precondition: At(x, y) ^ Mobile(x),
Effect: ¬At(x, y) ^ At(x, z))

Action( Load(m, x, y),
Precondition: At(m, y) ^ Package(y, x),
Effect: ¬Package(y, x) ^ Package(m, x) ^ At(m, y))

Action( Deliver(m, x, y),
Precondition: At(m, y) ^ Package(m, x),
Effect: ¬Package(m, x) ^ Package(y, x) ^ At(m, y))
\(At(Truck, UPSD) \land Package(UPSD, P1) \land Package(UPSD, P2) \land Mobile(Truck)\)
At(Truck, UPSD) \land Package(UPSD, P1) \\
\land Package(UPSD, P2) \land Mobile(Truck) \\

\textbf{Action( Load(m, x, y),} \\
\textbf{Precondition: At(m, y) \land Package(y, x),} \\
\textbf{Effect: \lnot Package(y, x) \land Package(m, x))}
\[
\text{Action}(\text{Load}(m, x, y), \text{Precondition: } \text{At}(m, y) \land \text{Package}(y, x), \text{Effect: } \neg \text{Package}(y, x) \land \neg \text{Package}(m, x))
\]
\( \text{At}(\text{Truck}, \text{UPSD}) \land \neg \text{Package}(\text{UPSD}, P1) \land \text{Package}(\text{UPSD}, P2) \land \text{Mobile}(\text{Truck}) \land \neg \text{Package}(\text{Truck}, P1) \land \text{Apply effects} \)

Action( Load(\text{Truck}, P1, \text{UPSD}) ),

Precondition: \( \text{At}(\text{Truck}, \text{UPSD}) \land \text{Package}(\text{UPSD}, P1) \),

Effect: \( \neg \text{Package}(\text{UPSD}, P1) \land \text{Package}(\text{Truck}, P1) \)
At(Truck, UPSD) \land Package(Truck, P1) \
\land Package(UPSD, P2) \land Mobile(Truck)

Action(\text{Load}(Truck, P2, UPSD),
Precondition: At(Truck, UPSD) \land Package(UPSD, P2),
Effect: \neg Package(UPSD, P2) \land Package(Truck, P2))
\( At(Truck, UPSD) \land Package(Truck, P1) \land Package(Truck, P2) \land Mobile(Truck) \)

Action( GoTo(Truck, UPSD, H1),
Precondition: \( At(Truck, UPSD) \land Mobile(Truck) \),
Effect: \( \neg At(Truck, UPSD) \land At(Truck, H1) \))
\[ \text{At(Truck, H1)} \land \text{Package(Truck, P1)} \land \text{Package(Truck, P2)} \land \text{Mobile(Truck)} \]

**Action**: \( \text{Deliver(Truck, P1, H1)} \),

**Precondition**: \( \text{At(Truck, H1)} \land \text{Package(Truck, P1)} \),

**Effect**: \( \neg \text{Package(Truck, P1)} \land \text{Package(H1, P1)} \)
\[\text{At(Truck, H1)} \land \text{Package(H1, P1)} \land \text{Package(Truck, P2)} \land \text{Mobile(Truck)}\]

Action( GoTo(Truck, H1, H2),
Precondition: \(\text{At(Truck, H1)} \land \text{Mobile(Truck)}\),
Effect: \(\neg \text{At(Truck, H1)} \land \text{At(Truck, H2)}\))
At(Truck, H2) ∧ Package(H1, P1) ∧ Package(Truck, P2) ∧ Mobile(Truck)

Action( Deliver(Truck, P2, H2),
Precondition: At(Truck, H2) ∧ Package(Truck, P2),
Effect: ¬Package(Truck, P2) ∧ Package(H2, P2))
At(Truck, H2) \land Package(H1, P1) \\
\land Package(H2, P2) \land Mobile(Truck)
Do I need the “Mobile()” at all?

Do I need separate actions for `Load()` and `Deliver()` or can I just have the “house load from truck”?

\begin{align*}
\text{Action}(&\ GoTo(x, y, z), \\
\text{Precondition: } &At(x, y) \land Mobile(x), \\
\text{Effect: } &\neg At(x, y) \land At(x, z))
\end{align*}

\begin{align*}
\text{Action}(&\ Load(m, x, y), \\
\text{Precondition: } &At(m, y) \land Package(y, x), \\
\text{Effect: } &\neg Package(y, x) \land Package(m, x) \land At(m, y))
\end{align*}

\begin{align*}
\text{Action}(&\ Deliver(m, x, y), \\
\text{Precondition: } &At(m, y) \land Package(m, x), \\
\text{Effect: } &\neg Package(m, x) \land Package(y, x) \land At(m, y))
\end{align*}
Forward search

While the solution might seem obvious to us, the search space is (surprisingly) quite large.

The brute force way (forward search) simply looks at all valid actions from the current state.

We can then search it in using BFS (or iterative deepening) to find fewest action cost goal.
Forward search

GoTo(Truck, USPD)

At(USPD)

At(H1)

At(H2)

At(USPD) ∧ Package(P1)

Load(USPD, P1, USPD)

can ignore
Forward search

Actions: 3 (Move, Deliver, Load)
Objects: 6 (Truck, USPD, H1, H2, P1, P2)
Min moves to goal: 6 (L, L, G, D, G, D)

Despite this problem being simplistic, the branching factor is about 4 to 5 (even with removing redundant actions)

This means we could search around 10,000 states before we found the goal
Forward search

This search is actually much more than the number of states due to redundant paths.

Package() can be: UPSD, Truck, H1, H2
At() can be: UPSD, Truck, H1, H2, P1, P2

There are 2 packages for Package()
There is 1 truck for At()

So total states = $4^2 * 6 = 96$
Like to backward chaining in first order logic, we can start at the goal state go backwards.

This helps reduce the number of redundant states we search (sorta), but this adds some complications (discuss in a bit).

As our actions are defined “going forwards” we have to apply the actions “in reverse” (or an inverse action: action$^{-1}()$)
Backward search

The book gives the full formal way to apply actions in reverse:

\[
\text{POS(new state)} = (\text{POS(old state)} - \text{ADD(action)}) \cup \text{POS(Precondition(action))} \\
\text{NEG(new state)} = (\text{NEG(old state)} - \text{DEL(action)}) \cup \text{NEG(Precondition(action))}
\]

... where POS() are the positive relations in a state (and NEG() is similarly negative)

ADD() are the relations that will be added by the action (and DEL() the relations that will be removed/deleted by the action)
Backward search

\[
\text{POS}(\text{new state}) = (\text{POS}(\text{old state}) - \text{ADD}(\text{action})) \cup \text{POS}(\text{Precondition}(\text{action}))
\]

\[
\text{NEG}(\text{new state}) = (\text{NEG}(\text{old state}) - \text{DEL}(\text{action})) \cup \text{NEG}(\text{Precondition}(\text{action}))
\]

So to do an action “backwards”:

1. Removing action effects (in reverse)
   All positive effect relations are removed
   If we are using negative relations, all
   negative effect relations are removed

2. Adding in precondition effects (pos&neg)

\[
\text{Action}(\, \text{Deliver}(m, x, y),
\]
Precondition: \( At(m, y) \land \text{Package}(m, x), \)
Effect: \( \neg \text{Package}(m, x) \land \text{Package}(y, x) \)
Backward search

POS(new state) = (POS(old state) - ADD(action)) ∪ POS(Precondition(action))
NEG(new state) = (NEG(old state) - DEL(action)) ∪ NEG(Precondition(action))

So to do an action “backwards”:

1. Removing **action effects** (in reverse)
   All **positive effect relations** are removed
   If we are using negative relations, all **negative effect relations** are removed

2. Adding in **precondition** effects (pos&neg)

**Action**( Deliver(m, x, y),

**Precondition**: At(m, y) ∧ Package(m, x),

**Effect**: ¬Package(m, x) ∧ Package(y, x))
Backward search

Action( Deliver(m, x, y),
Precondition: At(m, y) \land Package(m, x),
Effect: \neg Package(m, x) \land Package(y, x))

So if we started with: Package(H2,P2)
Substitute: y/H2, x/P2 (m can stay just “m”)
Remove positives: Package(H2,P2)
Remove negatives: (nothing to do as “start” state has no negatives, just Package(H2,P2))
Add precondition: At(m, H2) \land Package(m, P2)
At(Truck, H2) \land Package(H1, P1) \land Package(Truck, P2) \land Mobile(Truck)

Try to continue from here!

Action( Deliver(m, x, y),
Precondition: At(m, y) \land Package(m, x),
Effect: \neg Package(m, x) \land Package(y, x) \land At(m, y))

Action( Load(m, x, y),
Precondition: At(m, y) \land Package(y, x),
Effect: \neg Package(y, x) \land Package(m, x) \land At(m, y))

Action( GoTo(x, y, z),
Precondition: At(x, y) \land Mobile(x),
Effect: \neg At(x, y) \land At(x, z))