Lecture 4
Planning and Scheduling
Lecture Plan

1. Lab Assignment Discussion

2. Planning:
   I. PDDL
   II. Solving PDDL problems: A* & Planning Graphs

3. (Scheduling)
   I. Dynamic Programming for Scheduling
   II. Resource Constraints and the Minimum Slack Algorithm
PDDL: Planning Domain Definition Language

- States
- Actions
- Goal
PDDL: Planning Domain Definition Language

- **States**
  - States are specified by a set of atomic ‘fluents’ (statements that can be true or false).
  - Atomic means they do not include conjunctions, disjunctions, conditionals or negation.
  - Database semantics is assumed, allowing negation by omission.
    - Unique names assumption: Different names refer to different objects
    - Closed-world assumption: If not true, then false.
    - Domain closure: No unnamed objects.

- **Actions**

- **Goal**
PDDL: Planning Domain Definition Language

- States
- Actions
  - Specified by a set of preconditions and effects, both of which are conjunctions of ‘literals’ - atomic propositions or negations of atomic propositions. All preconditions are current, all effects immediate.
  - Actions can be performed at a state where all positive literals are, and all negative literals are not, in the set of fluents at a state.
  - The effect of an action is to add all positive literals, and remove all negative literals, in the action's effects from the current state.
- Goal
How actions function as transitions:

State 1: Set, $F$, of Atomic Fluents

State 2: Set, $G = F \cup E^+ \setminus A(E^-)$, of Atomic Fluents

Preconditions: Set, $P = P^+ \cup P^-$, of literals, where:
- $P^+ \subseteq F$
- $A(P^-) \cap F = \emptyset$

Effects: Set, $E = E^+ \cup E^-$, of literals.
PDDL: Planning Domain Definition Language

- States
- Actions
- Goal
  - Specified by a conjunction of literals.
  - We have achieved a goal state when all positive literals are, and all negative literals are not, included in the current state specification.
A PDDL Problem

1. An initial state

2. A set of actions:
   
   Action schema used to represent classes of actions using variables:
   
   Action: Eat Cake \((x,y)\)
   
   Precondition: Cake\((x)\), -Eaten\((x)\), Person\((y)\), Hungry\((y)\)
   
   Effect: Eaten\((x)\), -Hungry\((y)\)

3. A goal specification.
PDDL Problems as Search

1. We must discover the transitions we can make by seeing which actions can be performed.

2. It is easy to search backwards and discover states that can transition to our current state given a possible action.

   To find predecessor state to current state given a particular action: Remove all effects (add negations) and add all preconditions (remove negations) from current state.
PDDL Example: Initial State

Initial state:

Person (mike)
PDDL Example: Actions

Action: Go Shopping (x)
Preconditions: Person(x)
Effects: HasMix(x)
PDDL Example: Actions

<table>
<thead>
<tr>
<th>Action:</th>
<th>Make Cake(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions:</td>
<td>Person(x)</td>
</tr>
<tr>
<td></td>
<td>HasMix (x)</td>
</tr>
<tr>
<td>Effects:</td>
<td>HasCake(x)</td>
</tr>
<tr>
<td></td>
<td>-HasMix(x)</td>
</tr>
</tbody>
</table>
PDDL Example: Actions

Action: Wait(x)
Preconditions: Person(x), -Hungry(x)
Effects: Hungry(x)
PDDL Example: Actions

Action: Eat Cake A (x)

Preconditions: Person(x), Hungry(x), HasCake(x)

Effects: -HasCake(x), -Hungry(x), EatenCake(x)
### PDDL Example: Actions

<table>
<thead>
<tr>
<th>Action:</th>
<th>Eat Cake B (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions:</td>
<td>Person(x),</td>
</tr>
<tr>
<td></td>
<td>-Hungry(x),</td>
</tr>
<tr>
<td></td>
<td>HasCake(x)</td>
</tr>
<tr>
<td>Effects:</td>
<td>-HasCake(x),</td>
</tr>
<tr>
<td></td>
<td>-Person(x)</td>
</tr>
</tbody>
</table>
PDDL Example: Goal

Preconditions: Person(mike),
-Hungry(mike),
HasCake(mike),
EatenCake(mike)

We could have a goal specification that included variables. Eg Let anyone who is a person, not just mike, eat a cake while having a cake and not being hungry:

Person(x),
-Hungry(x),
HasCake(x),
EatenCake(x)
<table>
<thead>
<tr>
<th>Action:</th>
<th></th>
<th>Action:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Eat Cake B (x)</td>
<td>Pre.:</td>
<td>Person(x)</td>
<td>Pre.:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Hungry(x),</td>
<td>Person(x)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HasCake(x)</td>
<td>Hungry(x),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effects:</td>
<td>HasCake(x)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-HasCake(x)</td>
<td>EatenCake(x),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Person(x)</td>
<td>-Hungry(x)</td>
</tr>
<tr>
<td>Goal</td>
<td></td>
<td>Action:</td>
<td></td>
</tr>
<tr>
<td>Pre.:</td>
<td>Person(mike),</td>
<td>Eat Cake A (x)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-Hungry(mike),</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HasCake(mike),</td>
<td>Effects:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EatenCake(mike)</td>
<td>-HasCake(x),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Person(x)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Person(x)</td>
<td>Pre.:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Go Shopping (x)</td>
<td>Person(mike),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effects:</td>
<td>-Hungry(mike),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Person(x)</td>
<td>HasCake(mike),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action:</td>
<td>EatenCake(mike),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Person(x)</td>
<td>-Hungry(x)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Go Shopping (x)</td>
<td>Effects:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Person(x)</td>
<td>Hungry(x)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effects:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-HasMix(x)</td>
<td></td>
</tr>
<tr>
<td>Action:</td>
<td></td>
<td>Person(x)</td>
<td>Watch TV (x)</td>
</tr>
<tr>
<td>Pre.:</td>
<td>Person(mike),</td>
<td>Effects:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-Hungry(mike),</td>
<td>-HasCake(x),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HasCake(mike),</td>
<td>-Person(x)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EatenCake(mike)</td>
<td>Action:</td>
<td></td>
</tr>
<tr>
<td>Goal</td>
<td></td>
<td>Person(mike)</td>
<td>Go Shopping (x)</td>
</tr>
<tr>
<td>Pre.:</td>
<td>Person(mike),</td>
<td>Effects:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-Hungry(mike),</td>
<td>-HasCake(x),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HasCake(mike),</td>
<td>-Person(x)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EatenCake(mike)</td>
<td>Action:</td>
<td></td>
</tr>
<tr>
<td>Goal</td>
<td></td>
<td>Person(mike)</td>
<td>Go Shopping (x)</td>
</tr>
<tr>
<td>Pre.:</td>
<td>Person(mike),</td>
<td>Effects:</td>
<td></td>
</tr>
<tr>
<td>Initial State</td>
<td>Person(mike)</td>
<td>-Hungry(mike),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>HasCake(mike),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>EatenCake(mike)</td>
<td></td>
</tr>
<tr>
<td>Action:</td>
<td>Eat Cake A (x)</td>
<td>Eat Cake B (x)</td>
<td>Make Cake(x)</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------</td>
<td>----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Pre.:</td>
<td>Person(x)</td>
<td>Person(x)</td>
<td>Person(x)</td>
</tr>
<tr>
<td></td>
<td>Hungry(x), HasCake(x)</td>
<td>Hungry(x), HasCake(x)</td>
<td></td>
</tr>
<tr>
<td>Effects:</td>
<td>-HasCake(x), EatenCake(x), -Hungry(x)</td>
<td>-HasCake(x)</td>
<td>HasMix(x)</td>
</tr>
<tr>
<td></td>
<td>Person(x)</td>
<td>Person(x)</td>
<td>HasMix(x)</td>
</tr>
<tr>
<td>Goal</td>
<td>Person(mike), Hungry(mike), HasCake(mike), EatenCake(mike)</td>
<td>Person(mike), Hungry(mike), HasCake(mike)</td>
<td>HasMix(mike)</td>
</tr>
<tr>
<td>Action: <strong>Eat Cake B (x)</strong></td>
<td>Action: <strong>Eat Cake A (x)</strong></td>
<td>Action: <strong>Make Cake(x)</strong></td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------</td>
<td>---------------------------</td>
<td></td>
</tr>
<tr>
<td>Pre.: Person(x)</td>
<td>Pre.: Person(x)</td>
<td>Pre.: Person(x)</td>
<td></td>
</tr>
<tr>
<td>-Hungry(x), HasCake(x)</td>
<td>-Hungry(x), HasCake(x)</td>
<td>HasMix(x)</td>
<td></td>
</tr>
<tr>
<td>Effects: -HasCake(x), -Person(x)</td>
<td>Effects: -HasCake(x), EatenCake(x), -Hungry(x)</td>
<td>Effects: HasCake(x)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-HasMix(x)</td>
<td></td>
</tr>
<tr>
<td><strong>Goal</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre.: Person(mike), -Hungry(mike), HasCake(mike), EatenCake(mike)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action: <strong>Go Shopping (x)</strong></td>
<td></td>
<td>Action: <strong>Wait (x)</strong></td>
<td></td>
</tr>
<tr>
<td>Pre.: Person(x)</td>
<td>Pre.: Person(x)</td>
<td>Pre.: Person(x)</td>
<td></td>
</tr>
<tr>
<td>Effects: HasMix(x)</td>
<td></td>
<td>Effects: -Hungry(x)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Hungry(x)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Initial State</strong></td>
<td><strong>Satisfied by omission</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Person(mike)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action:</td>
<td>Person(mike)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Initial State</strong></td>
<td>Person(x)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-Hungry(x),</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HasCake(x)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Goal</strong></td>
<td>Person(mike),</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-Hungry(mike),</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HasCake(mike),</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EatenCake(mike)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unsatisfied</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action:</th>
<th>Person(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Eat Cake B (x)</strong></td>
<td>Person(x)</td>
</tr>
<tr>
<td>Pre.:</td>
<td>-Hungry(x),</td>
</tr>
<tr>
<td></td>
<td>HasCake(x)</td>
</tr>
<tr>
<td>Effects:</td>
<td>-HasCake(x)</td>
</tr>
<tr>
<td></td>
<td>-Person(x)</td>
</tr>
<tr>
<td><strong>Eat Cake A (x)</strong></td>
<td>Person(x)</td>
</tr>
<tr>
<td>Pre.:</td>
<td>Hungry(x),</td>
</tr>
<tr>
<td></td>
<td>HasCake(x)</td>
</tr>
<tr>
<td>Effects:</td>
<td>-HasCake(x),</td>
</tr>
<tr>
<td></td>
<td>EatenCake(x),</td>
</tr>
<tr>
<td></td>
<td>-Hungry(x)</td>
</tr>
<tr>
<td><strong>Make Cake(x)</strong></td>
<td>Person(x)</td>
</tr>
<tr>
<td>Pre.:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HasMix(x)</td>
</tr>
<tr>
<td>Effects:</td>
<td>HasCake(x)</td>
</tr>
<tr>
<td></td>
<td>-HasMix(x)</td>
</tr>
<tr>
<td><strong>Wait (x)</strong></td>
<td>Person(x)</td>
</tr>
<tr>
<td>Pre.:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-Hungry(x)</td>
</tr>
<tr>
<td>Effects:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action:</th>
<th>Person(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Go Shopping (x)</strong></td>
<td>Person(x)</td>
</tr>
<tr>
<td>Pre.:</td>
<td></td>
</tr>
<tr>
<td>Effects:</td>
<td>HasMix(x)</td>
</tr>
</tbody>
</table>

**Goal**

Person(mike), Hungry(mike), HasCake(mike), EatenCake(mike)
<table>
<thead>
<tr>
<th>Action: Eat Cake B (x)</th>
<th>Action: Eat Cake A (x)</th>
<th>Action: Make Cake(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre.: Person(x)</td>
<td>Pre.: Person(x)</td>
<td>Pre.: Person(x)</td>
</tr>
<tr>
<td>-Hungry(x),</td>
<td>Hungry(x),</td>
<td>HasMix(x)</td>
</tr>
<tr>
<td>HasCake(x)</td>
<td>HasCake(x)</td>
<td>HasCake(x)</td>
</tr>
<tr>
<td>Effects: -HasCake(x)</td>
<td>Effects: -HasCake(x),</td>
<td>Effects: HasMix(x)</td>
</tr>
<tr>
<td>-Person(x)</td>
<td>EatenCake(x),</td>
<td>-HasMix(x)</td>
</tr>
<tr>
<td></td>
<td>-Hungry(x)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action: Go Shopping (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre.: Person(x)</td>
</tr>
<tr>
<td>Effects: HasMix(x)</td>
</tr>
</tbody>
</table>

**Goal**

Pre.: Person(mike),
-Hungry(mike),
HasCake(mike),
EatenCake(mike)

**Initial State**

Person(mike)

**Unsatisfied**
**Action:** Eat Cake B (x)  
*Pre.:* Person(x)  
-Hungry(x),  
HasCake(x)  
*Effects:*  
-HasCake(x)  
-Person(x)

**Goal**  
*Pre.:* Person(mike),  
-Hungry(mike),  
HasCake(mike),  
EatenCake(mike)

**Action:** Eat Cake A (x)  
*Pre.:* Person(x)  
-Hungry(x),  
HasCake(x)  
*Effects:*  
-HasCake(x),  
EatenCake(x),  
-Hungry(x)

**Action:** Go Shopping (x)  
*Pre.:* Person(x)  
*Effects:* HasMix(x)

**Action:** Make Cake(x)  
*Pre.:* Person(x)  
*Effects:* HasMix(x)

**Action:** Wait (x)  
*Pre.:* Person(x)  
*Effects:* Hungry(x)

---

**Initial State**  
Person(mike)

**Unsatisfied**
Initial State

Person(mike)

Unsatisfied

Action: **Eat Cake B (x)**
Pre.: Person(x)
-Hungry(x),
HasCake(x)
Effects: -HasCake(x)
-Person(x)

Action: **Eat Cake A (x)**
Pre.: Person(x)
Hungry(x),
HasCake(x)
Effects: -HasCake(x),
EatenCake(x),
-Hungry(x)

Action: **Make Cake(x)**
Pre.: Person(x)
HasMix(x)
Effects: HasCake(x)
-HasMix(x)

Action: **Wait (x)**
Pre.: Person(x)
Effects: Hungry(x)

Action: **Go Shopping (x)**
Pre.: Person(x)
Effects: HasMix(x)

Goal
Pre.: Person(mike),
-Hungry(mike),
HasCake(mike),
EatenCake(mike)
**Initial State**

Person(mike)

**Goal**

Initial State

Person(mike)

Satisfied
<table>
<thead>
<tr>
<th>Action: Eat Cake B (x)</th>
<th>Action: Eat Cake A (x)</th>
<th>Action: Make Cake(x)</th>
<th>Action: Wait (x)</th>
<th>Action: Go Shopping (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre.: Person(x)</td>
<td>Pre.: Person(x)</td>
<td>Pre.: Person(x)</td>
<td>Pre.: Person(x)</td>
<td>Pre.: Person(x)</td>
</tr>
<tr>
<td></td>
<td>-Hungry(x)</td>
<td>Hungry(x),</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HasCake(x)</td>
<td>HasCake(x)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Effects:</td>
<td>-HasCake(x)</td>
<td>-HasCake(x),</td>
<td></td>
<td>HasMix(x)</td>
</tr>
<tr>
<td></td>
<td>-Person(x)</td>
<td>EatenCake(x),</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Hungry(x)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal</td>
<td></td>
<td>Effects:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre.: Person(mike)</td>
<td></td>
<td>Person(x)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-Hungry(mike)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HasCake(mike)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EatenCake(mike)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Satisfied (the second by omission).
**Initial State**

Person(mike), Person(mike),

**Goal**

Pre.: Person(mike), Hungry(mike), HasCake(mike), EatenCake(mike)

---

**Action:** Eat Cake B (x)

Pre.: Person(x)

Effects: -HasCake(x), -Person(x)

**Action:** Eat Cake A (x)

Pre.: Person(x)

Effects: -HasCake(x), EatenCake(x), -Hungry(x)

**Action:** Go Shopping (x)

Pre.: Person(x)

Effects: HasMix(x)

**Action:** Wait (x)

Pre.: Person(x)

Effects: Hungry(x)

**Action:** Make Cake(x)

Pre.: Person(x)

Effects: HasMix(x)

-HasMix(x)

**Action:** Go Shopping (x)

Pre.: Person(x)

Effects: HasMix(x)
At the next step, from S1 we could go shopping again. But it wouldn’t change anything.
At the next step, from S1 we could go shopping again. But it wouldn’t change anything.

The preconditions of the action are met...
At the next step, from S1 we could go shopping again. But it wouldn’t change anything.

The preconditions of the action are met...

But the effects are already true at this state.
Person(mike), Hungry(mike)

Initial State

Go Shopping (mike)

Wait (mike)

S1

S2

S3

Go Shopping (mike)

Make Cake (mike)

Action: Make Cake(x)
Pre.: Person(x)
HasMix(x)

Effects: HasCake(x)
-HasMix(x)

Or from S1 we could make a cake which would change things.
Initial State

Person(mike), Hungry(mike)

Go Shopping (mike)

Wait (mike)

S1

Person(mike), HasMix(mike)

Go Shopping (mike)

Make Cake (mike)

S2

S3

Person(mike), HasCake(mike)

Action: Make Cake(x)
Pre.:  Person(x)
       HasMix(x)

Effects: HasCake(x)
         -HasMix(x)

Or from S1 we could make a cake which would change things.

The preconditions of the action are met...
Person(mike), Hungry(mike)

Initial State

Go Shopping (mike)

Wait (mike)

S1

Person(mike), HasMix(mike)

Go Shopping (mike)

Make Cake (mike)

S2

Person(mike), Hungry(mike)

S3

Person(mike), HasMix(mike)

Action: Make Cake(x)
Pre.: Person(x)
     HasMix(x)
Effects: HasCake(x)
         -HasMix(x)

Or from S1 we could make a cake which would change things.

The preconditions of the action are met...

And the effects cause one fluent to be made true...
Person(mike), Hungry(mike)

Initial State

Go Shopping (mike)

S1

Go Shopping (mike)

Wait (mike)

S2

Make Cake (mike)

S3

Action: Make Cake(x)

Pre.: Person(x)

HasMix(x)

Effects: HasCake(x)

-HasMix(x)

Or from S1 we could make a cake which would change things.

Person(mike), HasMix(mike)

Person(mike), HasCake(mike)

Person(mike), Hungry(mike)

The preconditions of the action are met...

And the effects cause one fluent to be made true...

And one to be made false.
Initial State

Go Shopping (mike)

S1 -> Wait (mike)

Person(mike), HasMix(mike)

S2 -> Wait (mike)

Person(mike), Hungry(mike)

S1 -> Make Cake (mike)

Person(mike), HasMix(mike)

S3

S3 -> Wait (mike)

Person(mike), HasCake(mike)

S4

Action: Wait (x)
Pre.: Person(x)
    - Hungry(x)
Effects: Hungry(x)

Or we could wait.
From S2 we can only go shopping. And that will take us to S4 by another route.
From S3 we can go shopping again.
Or we can eat the cake with action ‘Eat Cake B’ (by omission Mike is not hungry). This leads to a state with no atomic fluents. Everything is negated. Including ‘Person(mike)’: Mike is dead in this state!
Initial State

Go Shopping (mike)

Go Shopping (mike)

Wait (mike)

Make Cake (mike)

Go Shopping (mike)

Go Shopping (mike)

Wait (mike)

Person(mike), HasCake(mike)

Or we can wait (and Mike lives...)

Action: Wait (x)

Pre.: Person(x)

-Hungry(x)

Effects: Hungry(x)
From S4 we can go shopping again, but that doesn’t change anything.

Action: Go Shopping (x)
Pre.: Person(x)
Effects: HasMix(x)
Initial State

Go Shopping (mike)

S1

Go Shopping (mike)

S2

Wait (mike)

S3

Wait (mike)

S4

Go Shopping (mike)

S5

Go Shopping (mike)

S6

Make Cake (mike)

S7

Eat Cake B (mike)

Person(mike), HasMix(mike), HasCake(mike)

Person(mike), HasMix(mike), Hungry(mike)

Action: Make Cake(x)

Pre.: Person(x)

Effects: HasCake(x)

Or we can make a cake, which takes us to S7
Let's compress the diagram...

Action: Make Cake(x)
Pre.: Person(x)

Effects: HasCake(x)
Initial State

Person(mike),
HasMix(mike),
HasCake(mike)

S5

S6

Lets compress the diagram...

Person(mike),
HasCake(mike),
Hungry(mike)
Initial State

S1, S2, S3, S4, S5, S6

Person(mike), HasMix(mike), HasCake(mike)

Action: Go Shopping (mike)
Pre.: Person(x)
Effects: HasMix(x)

From S5, we can go shopping, changing nothing...

Person(mike), HasCake(mike), Hungry(mike)
From S5, we can make a cake... Leading back to S3, since we cannot express that we have multiple cakes.
Or we can eat the cake, leading to another way to kill Mike.

Action: Eat Cake B (x)
Pre.: Person(x)
- Hungry(x)
HasCake(x)
Effects: - Person(x),
- HasCake(x)
Person(mike), HasCake(mike), Hungry(mike)

Make Cake (mike)

Go Shopping (mike)

Eat Cake B (mike)

Wait (mike)

Pre.: Person(x) -Hungry(x)

Effects: Hungry(x)

Or we can wait.

Action: Wait (x)
We can’t do anything from S6, since all actions require Person(mike).
From S7, we can go shopping, leading to S9.

**Action:** Go Shopping (x)

**Pre.:** Person(x)

**Effects:** HasMix(x)
Or we can eat the cake. This time we use ‘Eat Cake A’, since Mike is hungry.
So this time he lives! (Yay)
Initial State

Person(mike), HasCake(mike), Hungry(mike)

Action: Eat Cake A (x)
Pre.: Person(x)
Hungry(x)
HasCake(x)
Effects: Hungry(x)
-HasCake(x)

Let's clear out the diagram (and stop showing actions)...

Person(mike), HasMix(mike), HasCake(mike)

Eat Cake B (mike)

S1
S2
S3
S4
S5
S6
S7
S8
S9
S10

Person(mike), HasMix(mike)
Person(mike), HasMix(mike), HasCake(mike), Hungry(mike)
Person(mike), EatenCake(mike)
Person(mike), HasMix(mike), HasCake(mike), Hungry(mike)
Person(mike), HasMix(mike), HasCake(mike), Hungry(mike)
Person(mike), HasMix(mike), HasCake(mike), Hungry(mike)

Make Cake (mike)

Go Shopping (mike)
Wait (mike)
Eat Cake A (mike)
Person(mike), HasMix(mike), HasCake(mike)

Person(mike), HasMix(mike), HasCake(mike), Hungry(mike)

Person(mike), HasMix(mike), HasCake(mike), Hungry(mike)

Lets clear out the diagram (and stop showing actions)
Person(mike), HasMix(mike), HasCake(mike)

Person(mike), HasMix(mike), HasCake(mike), Hungry(mike)

Person(mike), HasMix(mike), HasCake(mike), Hungry(mike)

Person(mike), HasMix(mike), HasCake(mike), Hungry(mike)

S8 is another terminal node, since we killed off Mike.
Person(mike), HasMix(mike), HasCake(mike)

At S9, we can go shopping but it does nothing.

Person(mike), HasMix(mike), HasCake(mike), Hungry(mike)

Go Shopping (mike)

Person(mike), HasMix(mike), HasCake(mike)

Person(mike), EatenCake(mike)
Person(mike), HasCake(mike), Hungry(mike)

S5

Person(mike), HasMix(mike), HasCake(mike)

S8

HasMix(mike)

S9

Go Shopping (mike)

Make Cake (mike)

S7

Person(mike), HasCake(mike), Hungry(mike)

Or we can make another cake. But that takes us back to S7.

S10

Person(mike), EatenCake(mike)
Person(mike),
HasMix(mike),
HasCake(mike)

Person(mike),
HasMix(mike),
HasCake(mike),
Hungry(mike)

Person(mike),
EatenCake(mike)

Go Shopping
(mike)

Make Cake
(mike)

Or we can eat the cake using 'Eat Cake A'.
At S10, we can go shopping, which also leads to S11.
Person(mike), HasMix(mike), HasCake(mike)

Person(mike), HasMix(mike), HasCake(mike), Hungry(mike)

Person(mike), HasMix(mike), EatenCake(mike)

Person(mike), HasMix(mike), EatenCake(mike), Hungry(mike)

Person(mike), HasMix(mike), EatenCake(mike), Hungry(mike)

Person(mike), HasMix(mike), HasCake(mike)

Person(mike), HasMix(mike), EatenCake(mike), Hungry(mike)

Go Shopping (mike)

Eat Cake A (mike)

Wait (mike)

Or we can wait
Finally, from S11 we can make a cake and reach a state that satisfies the goal conditions!

Goal conditions:
- Person(mike)
- HasCake(mike), EatenCake(mike)
- Hungry(mike)
Finally, from S11 we can make a cake and reach a state that satisfies the goal conditions!

Goal conditions:
Person(mike),
HasCake(mike),
EatenCake(mike),
-Hungry(mike)

We can do a few more things from S12 that will take us back to other states, but let's ignore them.
Backtrack to find multiple paths/plans.

And we can backtrack though the graph (not the search tree, which we didn’t show) to find possible paths.
Points to note

- Naive searches are not going to scale.
- Limitations regarding expression
  - Really for deterministic domains.
  - Can have difficulties with disjunctions (numerics a case in point).
- Specifications of actions VERY important.
  - Can overcome expressive limitations
  - Can avoid pointless searching.
- Loops in the search space
- Orphan nodes in the search space
- Potentially multiple goal nodes.
- Potentially multiple paths from the initial state to a goal node.
  - Often due to the fact that we can perform the same steps in multiple orders.
Solving PDDL problems: A* & Planning Graphs
Planning Graphs

- Directed graphs, organized into alternating state and action levels:
  \[ s_0, a_0, s_1, a_1, \ldots, s_n. \]
- Each action at \( a_i \) is connected to its precondition at \( s_i \) and effects at \( s_{i+1} \).
- Trivial ’no-op’ actions are given for all literals. No-op action for literal \( P \):
  - Preconditions: \( P \)
  - Effects: \( P \)
- Tracks (some) mutual exclusion relations.
Mutual Exclusion

Level $a_i$ contains all actions that can occur at level $s_i$ and records mutual exclusion links between incompatible actions. Two actions are recorded as incompatible if:

- Inconsistent effects – one action’s effect negates an effect of the other.
- Interference – one action’s effect is the negation of a precondition of the other.
- Competing needs – a precondition for one action is marked as mutually exclusive to a precondition for the other.

Level $s_{i+1}$ contains all literals could result from any subset of compatible actions at level $a_i$ and records mutual exclusion links between incompatible literals. Two literals are recorded as incompatible if:

- Contradictory – One is the negation of the other.
- Inconsistent support – If each possible pair of actions that could produce the literals are mutually exclusive.

Note: Not all mutual exclusions are tracked. Literals can appear before they are actually possible!
Planning Graph

\[
S_0
\]

<table>
<thead>
<tr>
<th>Person(mike)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Hungry(mike)</td>
</tr>
<tr>
<td>-HasCake(mike)</td>
</tr>
<tr>
<td>-HasMix(mike)</td>
</tr>
<tr>
<td>-EatenCake(mike)</td>
</tr>
</tbody>
</table>

The first state layer has all the literals true in the initial state.
Planning Graph

<table>
<thead>
<tr>
<th>$S_0$</th>
<th>$S_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person(mike)</td>
<td>□</td>
</tr>
<tr>
<td>-Hungry(mike)</td>
<td>□</td>
</tr>
<tr>
<td>-HasCake(mike)</td>
<td>□</td>
</tr>
<tr>
<td>-HasMix(mike)</td>
<td>□</td>
</tr>
<tr>
<td>-EatenCake(mike)</td>
<td>□</td>
</tr>
</tbody>
</table>

Each action layer has all the actions that could be perform given the first state layer.

Actions point to their preconditions.

Trivial actions are represented by ’□’. 
Planning Graph

Each state layer now has all literals that are the result of any action in the previous action later. Each state points to the actions it is an effect of.

Trivial actions ensure that the state layers are monotonically increasing (we never loose states).
Planning Graph

And we enter mutex (mutual exclusion) relations.

Two actions are recorded as incompatible if:
- one action’s effect negates an effect of the other.
- one action’s effect is the negation of a precondition of the other.
- a precondition for one action is marked as exclusive to a precondition for the other.

<table>
<thead>
<tr>
<th>$S_0$</th>
<th>$A_0$</th>
<th>$S_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person(mike)</td>
<td>Go Shopping</td>
<td>Person(mike)</td>
</tr>
<tr>
<td>-Hungry(mike)</td>
<td>□</td>
<td>Hungry(mike)</td>
</tr>
<tr>
<td>-HasCake(mike)</td>
<td>□</td>
<td>-HasCake(mike)</td>
</tr>
<tr>
<td>-HasMix(mike)</td>
<td>□</td>
<td>-HasMix(mike)</td>
</tr>
<tr>
<td>-EatenCake(mike)</td>
<td>□</td>
<td>-EatenCake(mike)</td>
</tr>
</tbody>
</table>
And we enter mutex (mutual exclusion) relations.

Two actions are recorded as incompatible if:

- one action’s effect negates an effect of the other.
- one action’s effect is the negation of a precondition of the other.
- a precondition for one action is marked as exclusive to a precondition for the other.
## Planning Graph

<table>
<thead>
<tr>
<th>$S_0$</th>
<th>$A_0$</th>
<th>$S_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person(mike)</td>
<td>□</td>
<td>Wait</td>
</tr>
<tr>
<td>-Hungry(mike)</td>
<td>□</td>
<td>Hungry(mike)</td>
</tr>
<tr>
<td>-HasCake(mike)</td>
<td>□</td>
<td>-HasCake(mike)</td>
</tr>
<tr>
<td>-HasMix(mike)</td>
<td>□</td>
<td>-HasMix(mike)</td>
</tr>
<tr>
<td>-EatenCake(mike)</td>
<td>□</td>
<td>-EatenCake(mike)</td>
</tr>
</tbody>
</table>

And we enter mutex (mutual exclusion) relations.

Two states are recorded as incompatible if:

- One is the negation of the other.
- If each possible pair of actions that could produce the literals are mutually exclusive.
And we enter mutex (mutual exclusion) relations.

Two states are recorded as incompatible if:

- **One is the negation of the other.**
- If each possible pair of actions that could produce the literals are mutually exclusive.
And we proceed.

Note we can create the graph iteratively...

And at each stage the number of actions/literals increases...
Two actions are recorded as incompatible if:

- one action’s effect negates an effect of the other.
- one action’s effect is the negation of a precondition of the other.
- a precondition for one action is marked as exclusive to a precondition for the other.
Two actions are recorded as incompatible if:

- one action’s effect negates an effect of the other.
- one action’s effect is the negation of a precondition of the other.
- a precondition for one action is marked as exclusive to a precondition for the other.
Two states are recorded as incompatible if:

- **One is the negation of the other.**
- If each possible pair of actions that could produce the literals are mutually exclusive.
Two states are recorded as incompatible if:

- One is the negation of the other.
- If each possible pair of actions that could produce the literals are mutually exclusive.
Planning Graph

We continue in this iterative fashion until two consecutive levels are identical.

At this point we say the graph has ‘leveled off’ and terminate.
What use are Planning Graphs?

1. If any goal literal fails to appear in the final state level, then the problem is unsolvable.

2. The number of levels between a state and the presence of all goal literals can be used to give an admissable heuristic (for, e.g. A*).
   1. The max-level heuristic given the number of levels until all goal literals are in the graph.
   2. The set-level heuristic given the number of levels until all goal literals are in the graph AND there are no mutexs between any pair of goal literals.

   This is unsurprising: We have effectively relaxed constraint on the problem, which is a general way of obtaining heuristics.

3. We can obtain a plan directly of the Planning Graph using the Graph Planning Algorithm, but this is generally less efficient.
Graph Planning Algorithm

From current state repeat until termination:

1. While at least one goal literals is absent OR there exists at least one mutex between goal literals:
   Create the next action and state levels, $A_c$ and $S_{c+1}$ and then set current state to $S_{c+1}$.

2. Otherwise, run the GP-Termination algorithm to see if the algorithm terminated. If it does not, create the next action and state levels, $A_c$ and $S_{c+1}$. 

GP-Termination Algorithm

1. Examine the last level, $S_i$, of the planning graph and initialize a list of literals, $L_i$, to be satisfied from the goal state of the planning problem.

2. Transition to layer $S_{i-1}$ through selecting any subset of the actions at layer $A_{i-1}$ where:
   1. These actions are not mutually exclusive
   2. The preconditions of these actions are not mutually exclusive.
   3. These actions cause as effects the list attached to the state. (Remember there are trivial actions.)

3. Set $L_{i-1}$ as the set of preconditions for actions involved in the transition.

4. If a state is found with a goal list that is a subset of the initial state, a solution path has been found and we terminate with success.

   **REQUIRES SEARCHING ALL SEQUENCES OF VALID SUBSETS OF ACTIONS AT EACH LAYER**

5. If no such state exists, no solution is currently possible and we record which goals were unsatisfiable at the final step.

6. If $S_c = S_{c-1}$ AND the goals that were unsatisfiable at $S_c$ are the same as those that were unsatisfiable at $S_{c-1}$ then we terminate with failure.