Announcements

- P3 out!
  - Sorry it is late. Due 3/7.
  - Can submit 3/8 without penalty
  - Can submit 3/9 with 1 late day
  - No submissions later than 3/9

- HW6 out soon!
Planning, Thus Far

Goal States
completely specified

Goal Statements
partially specified

Preference models
objective function

Increasing Generality

Robot Block Stacking

Start state: A, B, C on table
Goal: Block B on C and C on A
Actions: ?
Modeling Block Stacking States

Start state: A, B, C on table
Goal: Block B on C and C on A
Actions: ?

Goals in the World

Goal States
- completely specified

Goal Statements
- partially specified

Preference models
- objective function

Increasing Generality
Block Stacking States
Block Stacking States

States are Informationless

S1

S2

S3

S4

S5

S6

S7

S8

S9

S10

S11

S12

S13

S14

S15

S16

S17

S18

S19

S20

S21

S22
Initial and Goal States

Plan from Initial to Goal State
Goals in the World

Goal States
completely specified

Goal Statements
partially specified

Preference models
objective function

BFS, DFS, A*
Increasing Generality
Goals in the World

Goal States
- completely specified

Goal Statements
- partially specified

Preference models
- objective function

Increasing Generality
- BFS, DFS, A*
- CSP, LP, IP

CSP for Blocks World

Goal: Block B on C and C on A

Constraint Satisfaction Problem:
- Height(B) > Height(C)
- Height(C) > Height(A)
Goals in the World

Goal States
completely specified

Goal Statements
partially specified

Preference models
objective function

Increasing Generality
BFS, DFS, A*
CSP, LP, IP
Goals in the World

- **Goal States**
  - completely specified

- **Goal Statements**
  - partially specified

- **Preference models**
  - objective function

<table>
<thead>
<tr>
<th>Increasing Generality</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS, DFS, A*</td>
<td>CSP, LP, IP</td>
</tr>
</tbody>
</table>

Logical Agents

- Create a Knowledge Base
  - Symbols
  - Implications
Logical Agents

Create a Knowledge Base
Symbols – each is true or false
Implications – conjunctions imply new info

Symbolic Descriptions

Every state has the same objects
Properties and relationships of those objects change (i.e., locations)

Define states as set of symbols that represent whether those properties are true
Logical Agents

Create a Knowledge Base
Symbols

A
B
C
Robot Arm

Logical Agents

Create a Knowledge Base
Symbols

A-on-Table  A-In-Hand
B-on-Table  B-In-Hand
C-on-Table  C-In-Hand
Hand-Empty
A-on-B  B-on-A
A-on-C  C-on-A
B-on-C  C-on-B
Robot Arm

A
B
C
Logical Agents

Create a Knowledge Base

Symbols

<table>
<thead>
<tr>
<th>A-on-Table</th>
<th>A-in-Hand</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-on-Table</td>
<td>B-in-Hand</td>
</tr>
<tr>
<td>C-on-Table</td>
<td>C-in-Hand</td>
</tr>
</tbody>
</table>

Hand-Empty

<table>
<thead>
<tr>
<th>A-on-B</th>
<th>B-on-A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-on-C</td>
<td>C-on-A</td>
</tr>
<tr>
<td>B-on-C</td>
<td>C-on-B</td>
</tr>
</tbody>
</table>

Logical Agents

Create a Knowledge Base

Implications

What are the implications of A-on-Table[t]?
Logical Agents

Create a Knowledge Base
Implications

Hand-Empty[t] AND A-on-Table[t] ↔
A-In-Hand[t-1] AND
Hand-Empty[t-2] AND
(A-on-B[t-2] OR A-on-C[t-2])

Is this enough?
Logical Agents

Create a Knowledge Base

Implications

Hand-Empty[t] AND A-on-Table[t] $\iff$


Is this enough? Need to specify !Hand-Empty[t-1]…

Logical Agents

Create a Knowledge Base

Symbols

Implications

Check whether the KB entails a query

Query – a subset of symbols in model
Logical Agents

Create a Knowledge Base
   Symbols
   Implications

Check whether the KB entails a query
   Query – a subset of symbols in model

   B-on-C[t] AND C-on-A[t] AND HandEmpty[t]?

Partially-Specified

We didn’t tell you what all goal symbols needed to be, only some

There are potentially many goal states that could satisfy this goal

We only need to search for one satisfying assignment of variables
Challenges of Logic Planning

We need symbols for each time step

The actions (e.g., picking a block) are implicitly represented as the implications

Easy to incorrectly specify an implication

So many symbols means it is hard to debug

Classical Planning

Also partially-specified

Create a Knowledge Base

- Symbols
- Predicates
- Implications
- Operators

Check whether the KB entails a query

A goal state (conjunction of predicates)
<table>
<thead>
<tr>
<th>Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Can be symbols</strong></td>
</tr>
<tr>
<td>A-on-Table, A-In-Hand,</td>
</tr>
<tr>
<td>B-on-Table, B-In-Hand,</td>
</tr>
<tr>
<td>C-on-Table, C-In-Hand,</td>
</tr>
<tr>
<td>Hand-Empty,</td>
</tr>
<tr>
<td>A-on-B, B-on-A,</td>
</tr>
<tr>
<td>A-on-C, C-on-A,</td>
</tr>
<tr>
<td>B-on-C, C-on-B</td>
</tr>
<tr>
<td>Clear-A, Clear-B,</td>
</tr>
<tr>
<td>Clear-C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Can be symbols</strong></td>
</tr>
<tr>
<td>A-on-Table, A-In-Hand,</td>
</tr>
<tr>
<td>B-on-Table, B-In-Hand,</td>
</tr>
<tr>
<td>C-on-Table, C-In-Hand,</td>
</tr>
<tr>
<td>Hand-Empty,</td>
</tr>
<tr>
<td>A-on-B, B-on-A,</td>
</tr>
<tr>
<td>A-on-C, C-on-A,</td>
</tr>
<tr>
<td>B-on-C, C-on-B</td>
</tr>
<tr>
<td>Clear-A, Clear-B,</td>
</tr>
<tr>
<td>Clear-C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Can be functional</strong></td>
</tr>
<tr>
<td>Instances: A, B, C</td>
</tr>
<tr>
<td>Propositions:</td>
</tr>
<tr>
<td>In-Hand(A)</td>
</tr>
<tr>
<td>On-Table(B)</td>
</tr>
<tr>
<td>On-Block(B,C)</td>
</tr>
<tr>
<td>HandEmpty()</td>
</tr>
<tr>
<td>Clear(A)</td>
</tr>
</tbody>
</table>
Predicates

Are there other functions that we could use?

YES!!

Your challenge is finding a representation that works for your environment
Instances: A, B, C
Propositions:
1) In-Hand(A)
2) In-Hand(B)
3) In-Hand(C)
4) On-Table(A)
5) On-Table(B)
6) On-Table(C)
7) On-Block(B,C)
8) On-Block(A,B)
9) HandEmpty()
Classical Planning

Also partially-specified

Create a Knowledge Base
Symbols Predicates
Implications Operators

Check whether the KB entails a query
A goal state (conjunction of predicates)

Models of Operators (Actions)

Actions can be applied only if all conditions are met
Actions change the state of the world
Represented as effects that add/delete predicates

Unlike the implications, we can represent actions with symbolic descriptions (e.g., pick-up, put-down)
Block Stacking States – Conjunctions of Predicates

Block Stacking Operators (Actions)
What actions are represented? What are the rules for applying actions?

Actions for Block Stacking

Blocks are picked up and put down by the hand
Blocks can be picked up only if they are clear
Hand can pick up a block only if the hand is empty
Hand can put down blocks on blocks or on the table
Pick Up Block from Table Example

Preconditions

HandEmpty
On-Table(b)
Clear(b)
Pick Up Block from Table Example

Preconditions
- HandEmpty
- On-Table(b)
- Clear(b)

Effects?

Pick Up Block from Table Example

Preconditions
- HandEmpty
- On-Table(b)
- Clear(b)

Effects
- Add: Holding(b)
- Delete: On-Table(b)
- HandEmpty
Pick Block from Block Example

Operators for Block Stacking

Pickup_from_Table(b):
  Pre: HandEmpty, Clear(b), On-Table(b)
  Add: Holding(b)
  Delete: HandEmpty, On-Table(b)

Pickup_from_Block(b,c):
  Pre: HandEmpty, On(b,c), b! = c
  Add: Holding(b), Clear(c)
  Delete: HandEmpty, On(b,c)
Operators for Block Stacking

Pickup_from_Table(b):
  Pre: HandEmpty, Clear(b), On-Table(b)
  Add: Holding(b)
  Delete: HandEmpty, On-Table(b)

Putdown_on_Table(b):
  Pre: Holding(b)
  Add: HandEmpty, On-Table(b)
  Delete: Holding(b)

Pickup_from_Block(b,c):
  Pre: HandEmpty, On(b,c), b!=c
  Add: Holding(b), Clear(c)
  Delete: HandEmpty, On(b,c)

Putdown_on_Block(b,c):
  Pre: Holding(b), Clear(c)
  Add: HandEmpty, On(b,c)
  Delete: Clear(c), Holding(b)

Example Plan of Actions

HandEmpty & On-Table(R) & On(T,R) & Clear(T) & On-Table(O) & Clear(O)
Example Plan of Actions

HandEmpty & On-Table(R) & On(T,R) & Clear(T) & On-Table(O) & Clear(O)

Pickup_from_Block(b,c):
Pre: HandEmpty, On(b,c), b!=c
Add: Holding(b), Clear(c)
Delete: HandEmpty, On(b,c)

Pickup_from_Table(b):
Pre: HandEmpty, Clear(b), On-Table(b)
Add: Holding(b)
Delete: HandEmpty, On-Table(b)
Example Plan of Actions

HandEmpty & On-Table(R) & On(T,R) & Clear(T) & On-Table(O) & Clear(O)
Pickup_from_Block(T,R)

Pickup_from_Block(b,c):
Pre: HandEmpty, On(b,c), Clear(c), b!=c
Add: Holding(b), Clear(c)
Delete: HandEmpty, On(b,c)

Example Plan of Actions

HandEmpty & On-Table(R) & On(T,R) & Clear(T) & On-Table(O) & Clear(O)
Pickup_from_Block(T,R)
On-Table(R) & Clear(T) & On-Table(O) & Clear(O)

Pickup_from_Block(b,c):
Pre: HandEmpty, On(b,c), Clear(c), b!=c
Add: Holding(b), Clear(c)
Delete: HandEmpty, On(b,c)
Example Plan of Actions

HandEmpty & On-Table(R) & On(T,R) & Clear(T) & On-Table(O) & Clear(O)
Pickup_from_Block(T,R)
On-Table(R) & Clear(T) & On-Table(O) & Clear(O) & Holding(T) & Clear(R)

Pickup_from_Block(b,c):
Pre: HandEmpty, On(b,c), Clear(c), b!=c
Add: Holding(b), Clear(c)
Delete: HandEmpty, On(b,c)
Example Plan of Actions

HandEmpty & On-Table(R) & On(T,R) & Clear(T) & On-Table(O) & Clear(O)

Pickup_from_Block(T,R)

On-Table(R) & Clear(T) & On-Table(O) & Clear(O) & Holding(T) & Clear(R)

Putdown_on_Table(T)

Putdown_on_Table(b):
Pre: Holding(b)
Add: HandEmpty, On-Table(b)
Delete: Holding(b)

Example Plan of Actions

HandEmpty & On-Table(R) & On(T,R) & Clear(T) & On-Table(O) & Clear(O)

Pickup_from_Block(T,R)

On-Table(R) & Clear(T) & On-Table(O) & Clear(O) & Holding(T) & Clear(R)

Putdown_on_Table(T)

On-Table(R) & Clear(T) & On-Table(O) & Clear(O) & Clear(R)

Putdown_on_Table(b):
Pre: Holding(b)
Add: HandEmpty, On-Table(b)
Delete: Holding(b)
Example Plan of Actions

HandEmpty & On-Table(R) & On(T,R) & Clear(T) & On-Table(O) & Clear(O)

Pickup_from_Block(T,R)

On-Table(R) & Clear(T) & On-Table(O) & Clear(O) & Holding(T) & Clear(R)

Putdown_on_Table(T)

On-Table(R) & Clear(T) & On-Table(O) & Clear(O) & Clear(R) & HandEmpty & On-Table(T)

Putdown_on_Table(b):
Pre: Holding(b)
Add: HandEmpty, On-Table(b)
Delete: Holding(b)

Example Plan of Actions

HandEmpty & On-Table(R) & On(T,R) & Clear(T) & On-Table(O) & Clear(O)

Pickup_from_Block(T,R)

On-Table(R) & Clear(T) & On-Table(O) & Clear(O) & Holding(T) & Clear(R)

Putdown_on_Table(T)

On-Table(R) & Clear(T) & On-Table(O) & Clear(O) & Clear(R) & HandEmpty & On-Table(T)
Example Plan of Actions

HandEmpty & On-Table(R) & On(T,R) & Clear(T) & On-Table(O) & Clear(O)

*Pickup_from_Block(T,R)*

On-Table(R) & Clear(T) & On-Table(O) & Clear(O) & Holding(T) & Clear(R)

*Putdown_on_Table(T)*

On-Table(R) & Clear(T) & *On-Table(O)* & Clear(O) & Clear(R) & *HandEmpty* & On-Table(T)

*Pickup_from_Table(O)*

Pickup_from_Table(b):

Pre: HandEmpty, Clear(b), On-Table(b)
Add: Holding(b)
Delete: HandEmpty, On-Table(b)
Example Plan of Actions

HandEmpty & On-Table(R) & On(T,R) & Clear(T) & On-Table(O) & Clear(O)

Pickup_from_Block(T,R)
On-Table(R) & Clear(T) & On-Table(O) & Clear(O) & Holding(T) & Clear(R)

Putdown_on_Table(T)
On-Table(R) & Clear(T) & On-Table(O) & Clear(O) & Clear(R) & HandEmpty & On-Table(T)

Pickup_from_Table(O)
On-Table(R) & Clear(T) & Clear(O) & Clear(R) & On-Table(T) & Holding(O)

Putdown_on_Block(O,R)
On-Table(R) & Clear(T) & Clear(O) & On-Table(T) & On(O,R) & HandEmpty

Properties of Planners

**Soundness**
- A planning algorithm is *sound* if all solutions found are legal plans
  - All preconditions and goals are satisfied
  - No constraints are violated (temporal, variable binding)

**Completeness**
- A planning algorithm is *complete* if a solution can be found whenever one actually exists
- A planning algorithm is *strictly complete* if all solutions are included in the search space

**Optimality**
- A planning algorithm is *optimal* if the order in which solutions are found is consistent with some measure of plan quality
Techniques for Planning

BFS – Find shortest action sequence
Reachability Graph Representation

States: Conjunctions of Predicates
Arrows = Actions
Every state in level K is \textit{reachable} in K actions
Space complexity in terms of \# predicates \( p \)?

\[ 2^p \]
Poll: Is BFS sound, complete, optimal?

Soundness - all solutions found are legal plans
Completeness - a solution can be found whenever one actually exists
Optimality - the order in which solutions are found is consistent with some measure of plan quality

Linear Planning

Idea: Since we have a conjunction of goal predicates, let’s try to solve one at a time

• Maintain a stack of achievable goals
• Use BFS (or anything else) to find a plan to achieve that single goal
• Add a goal back on the stack if a later change makes it violated
Linear Planning Example

Goal Stack:       Action Plan:
On-Table(T)      
On-Table(R)      
On(O,R)          
Clear(O)         
Clear(T)         

Linear Planning Example

Goal Stack:       Action Plan:
On-Table(T)      
On-Table(R)      
On(O,R)          
Clear(O)         
Clear(T)         

Linear Planning Example

Goal Stack:       Action Plan:
On-Table(T)      
On-Table(R)      
On(O,R)          
Clear(O)         
Clear(T)         

Linear Planning Example

Goal Stack:       Action Plan:
On-Table(T)      
On-Table(R)      
On(O,R)          
Clear(O)         
Clear(T)         

Linear Planning Example

Goal Stack:
- On-Table(R)
- On(O,R)
- Clear(O)
- Clear(T)

Action Plan:
- On-Table(T)
- Pickup(T)
- Put-Table(T)

Linear Planning Example

Goal Stack:
- On-Table(R)
- On(O,R)
- Clear(O)
- Clear(T)

Action Plan:
- On-Table(T)
- Pickup(T)
- Put-Table(T)
Linear Planning Example

Goal Stack:          Action Plan:
On(O,R)             On-Table(T)
Clear(O)            Pickup(T)
Clear(T)            Put-Table(T)
On-Table(R)

Linear Planning Example

Goal Stack:          Action Plan:
On(O,R)             On-Table(T)
Clear(O)            Pickup(T)
Clear(T)            Put-Table(T)
On-Table(R)
Linear Planning Example

Goal Stack: Clear(O) Clear(T)
Action Plan: On-Table(T) Pickup(T) Put-Table(T) On-Table(R) On(O,R) Pickup(O) Put(O,R)

Linear Planning Example

Goal Stack: Clear(O) Clear(T)
Action Plan: On-Table(T) Pickup(T) Put-Table(T) On-Table(R) On(O,R) Pickup(O) Put(O,R)
Linear Planning Example

Goal Stack: Action Plan:
On-Table(T) On-Table(T)
Pickup(T) Put-Table(T)
On-Table(R) On(O,R)
On(O,R) Pickup(O)
Put(O,R)
Clear(O)
Clear(T)

Linear Planning Example 2

Goal Stack: Action Plan:
On(R,O) On(R,O)
On-Table(T) On-Table(T)
On(O,T) On(O,T)
Clear(R) Clear(R)
Linear Planning Example 2

Goal Stack: Action Plan:
On(R,O) On(R,O)
On-Table(T) Pickup(T)
On(O,T) Put-Table(T)
Clear(R) Pickup(R)

Linear Planning Example 2

Goal Stack: Action Plan:
On-Table(T) On(R,O)
On(O,T) Pickup(T)
Clear(R) Put-Table(T)

Pickup(R)
Put(R,O)
Linear Planning Example 2

Goal Stack:
- On-Table(T)
- On(O,T)
- Clear(R)

Action Plan:
- On(R,O)
- Pickup(T)
- Put-Table(T)
- Pickup(R)
- Put(R,O)

Linear Planning Example 2

Goal Stack:
- On(O,T)
- Clear(R)

Action Plan:
- On(R,O)
- Pickup(T)
- Put-Table(T)
- Pickup(R)
- Put(R,O)
- On-Table(T)
Linear Planning Example 2

Goal Stack:
- On(O,T)
- Clear(R)

Action Plan:
- On(R,O)
  - Pickup(T)
  - Put-Table(T)
  - Pickup(R)
  - Put(R,O)
  - On-Table(T)

Linear Planning Example 2

Goal Stack:
- Clear(R)

Action Plan:
- On(R,O)
  - Pickup(T)
  - Put-Table(T)
  - Pickup(R)
  - Put(R,O)
  - On-Table(T)

Goal Stack:
- Clear(R)

Action Plan:
- On(R,O)
  - Pickup(T)
  - Put-Table(T)
  - Pickup(R)
  - Put(R,O)
  - On-Table(T)

Goal Stack:
- Clear(R)

Action Plan:
- On(R,O)
  - Pickup(T)
  - Put-Table(T)
  - Pickup(R)
  - Put(R,O)
  - On-Table(T)

Goal Stack:
- Clear(R)

Action Plan:
- On(R,O)
  - Pickup(T)
  - Put-Table(T)
  - Pickup(R)
  - Put(R,O)
  - On-Table(T)

Goal Stack:
- Clear(R)

Action Plan:
- On(R,O)
  - Pickup(T)
  - Put-Table(T)
  - Pickup(R)
  - Put(R,O)
  - On-Table(T)

Goal Stack:
- Clear(R)

Action Plan:
- On(R,O)
  - Pickup(T)
  - Put-Table(T)
  - Pickup(R)
  - Put(R,O)
  - On-Table(T)
Linear Planning Example 2

Goal Stack:
- On(R,O)
- Clear(R)

Action Plan:
- On(R,O)
- Pickup(T)
- Put-Table(T)
- Pickup(R)
- Put(R,O)
- On-Table(T)
- On(O,T)
- Pickup(R)
- Put-Table(R)
- Pickup(O)
- Put(O,T)

Linear Planning Example 2

Goal Stack:
- Clear(R)

Action Plan:
- On(R,O)
- Pickup(T)
- Put-Table(T)
- Pickup(R)
- Put(R,O)
- On-Table(T)
- On(O,T)
- Pickup(R)
- Put-Table(R)
- Pickup(O)
- Put(O,T)

Action Plan (cont’d):
- On(R,O)
- Pickup(R)
- Put(R,O)
Linear Planning Example 2

Goal Stack:
On(R,O)
Pickup(T)
Put-Table(T)
Pickup(R)
Put(R,O)
On-Table(T)
On(O,T)
Pickup(R)
Put-Table(R)
Pickup(O)
Put(O,T)

Action Plan:
On(R,O)
Pickup(R)
Put(R,O)
Clear(R)

Action Plan (cont’d):
On(R,O)
Pickup(R)
Put(R,O)
Clear(R)

What happened?
Is linear planning sound?
Is linear planning complete?
Is linear planning optimal?
Sussman’s Anomaly

A weakness of linear planning is that sometimes you get really long plans. One goal can be achieved. The second goal immediately undoes it.

Note: This isn’t just a choice of goals. The anomaly happens no matter which goal is first.

Linear Planning Example 2

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>On(R,O)</td>
<td>Pickup(T)</td>
<td>On(R,O)</td>
</tr>
<tr>
<td></td>
<td>Put-Table(T)</td>
<td>Pickup(R)</td>
</tr>
<tr>
<td></td>
<td>Pickup(R)</td>
<td>Put(R,O)</td>
</tr>
<tr>
<td></td>
<td>Put(R,O)</td>
<td>Clear(R)</td>
</tr>
<tr>
<td>On-Table(T)</td>
<td>On(O,T)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pickup(R)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Put-Table(R)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pickup(O)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Put(O,T)</td>
<td></td>
</tr>
</tbody>
</table>

What happened?
- Is linear planning sound? Yes
- Is linear planning complete? No
- Is linear planning optimal? No
Non-Linear Planning

Idea: Interleave goals to achieve plans
- Maintain a set of unachieved goals
- Search all interleavings of goals
- Add a goal back to the set if a later change makes it violated

Non-Linear Planning (Example 2)

Goal Set: Action Plan:
On(R,O) On(R,O)
On-Table(T) On-Table(T)
On(O,T) On(O,T)
Clear(R) Clear(R)
Non-Linear Planning (Example 2)

Goal Set: Action Plan:
On(R,O) On(R,O)
On-Table(T) Pickup(T)
On(O,T) Put-Table(T)
Clear(R) STOP-SWITCH-GOALS
Non-Linear Planning (Example 2)

Goal Set:
- On(R,O)
- On-Table(T)
- On(O,T)
- Clear(R)

Action Plan:
- On(R,O) Pickup(T)
- Put-Table(T)
- STOP-SWITCH-GOALS
- On(O,T) Pickup(O)
- Put(O,T)
- On(R,O) Pickup(R)
- Put(R,O)

6 vs 10 actions
Non-Linear Planning

Idea: Interleave goals to achieve plans
• Maintain a set of unachieved goals
• Search all interleavings of goals
• Add a goal back to the set if a later change makes it violated
• It is complete, but takes longer to search
• It can produce shorter plans
  – Optimal plans if all interleavings are searched

Reachability Graph Representation

States: Conjunctions of Predicates
Arrows = Actions
Every state in level K is reachable in K actions
Space complexity in terms of # predicates p?
Planning Graph Representation

Key idea: Construct an approximation of the reachability graph in polynomial space
- The planning graph computes the **possibly reachable** states although they aren’t necessarily feasible

Planning graphs contain two types of layers
- Proposition layers – all reachable predicates
- Action layers – actions that could be taken
Both layers represent one time step
GraphPlan

Two alternating stages:
- **Extend**: One time step (two layers) in the planning graph
- **Search**: Find a valid plan in the planning graph

GraphPlan finds a plan or proves that no plan has fewer time steps
- Each time step can contain multiple actions

Building a Planning Graph

Start the planning graph with all starting predicates

- HandEmpty
- On-Table(R)
- On-Table(O)
- On(T,R)
- Clear(T)
- Clear(O)
Building a Planning Graph

Extend the graph with all applicable actions, noops
Designate all effects (add/delete)

HandEmpty
On-Table(R)
On-Table(O)
On(T,R)
Clear(T)
Clear(O)

Pickup(T)
Pickup(O)

On-Table(R)
On-Table(O)
On(T,R)
Clear(T)
Clear(O)
Clear(R)

HandEmpty
On-Table(R)
On-Table(O)
On(T,R)
Clear(T)
Clear(O)
Clear(R)

Holding(T)
Holding(O)

HandEmpty
On-Table(R)
On-Table(O)
On(T,R)
Clear(T)
Clear(O)
Clear(R)

Holding(T)
Holding(O)
Building a Planning Graph

Extend the graph

Determine exclusive actions

Actions A and B are exclusive (mutex) at action-level $i$, if:

- **Interference**: one action deletes a precondition of the other
- **Inconsistency**: one action is the negation of the other
- **Competing Needs**: $p$ is a precondition of A and $q$ is a precondition of B, and $p$ and $q$ are exclusive in proposition-level $i-1$

Building a Planning Graph

Extend the graph with all applicable actions, noops

Determine exclusive actions

Actions A and B are exclusive (mutex) at action-level $i$, if:

- **Interference**: one action deletes a precondition of the other
- **Inconsistency**: one action is the negation of the other
- **Competing Needs**: $p$ is a precondition of A and $q$ is a precondition of B, and $p$ and $q$ are exclusive in proposition-level $i-1$
Building a Planning Graph

Extend the graph with all applicable actions, noops

Determine exclusive actions

Actions A and B are exclusive (mutex) at action-level $i$, if:

- Interference: one action deletes a precondition of the other
- Inconsistency: one action is the negation of the other
- Competing Needs: $p$ is a precondition of A and $q$ is a precondition of B, and $p$ and $q$ are exclusive in proposition-level $i - 1$
Search the Planning Graph

Extend until first time all goals are present at a proposition level
  – May not be a solution, at that level
For each goal at level \( i \) (in some arbitrary order)
  – Select an action at level \( i-1 \) that achieves that goal and is not exclusive with any other action already selected at that level
  – Add all its preconditions to the set of goals at level \( i-2 \)
  – Do this for all the goals at level \( i \)
    • Use already selected actions, when possible
  – Backtrack if no non-exclusive action exists

If search is exhausted, extend planning graph one more proposition level

Extend a Level

Extend the graph with all applicable actions, noops
Determine exclusive actions, propositions
Building a Planning Graph

Extend the graph with all applicable actions, noops
Determine exclusive actions, propositions

HandEmpty
On-Table(R)
On-Table(O)
On(T,R)
Clear(T)
Clear(O)
Pickup(T)
Pickup(O)
Holding(T)
Holding(O)

HandEmpty
On-Table(R)
On-Table(O)
On(T,R)
Clear(T)
Clear(O)
Pickup(O)
Put-Table(O)
Put(O,T)
Pickup(R)
Holding(T)
Holding(O)
Building a Planning Graph

Extend the graph with all applicable actions, noops
Determine exclusive actions, propositions

HandEmpty
On-Table(R)
On-Table(O)
On(T,R)
Clear(T)
Clear(O)

Pickup(T)
Put(T,R)
Put(T,O)
Put-Table(T)
Pickup(O)
Put-Table(O)
O
Pickup(R)
Put(O,T)
Pickup(R)

Holding(T)
Holding(O)

Clear(T)
Clear(O)
Clear(R)
Holding(T)
Holding(O)

ALL LEVEL 2 ACTIONS MUTALLY EXCLUSIVE...
WHAT KIND?

Pickup(T) + Pickup(O) + Pickup(R)
Put(T,R) + Put(T,O) + Put(T,U)
Put-Table(T) + Put-Table(O) + Put-Table(U)
Pickup(O) + Pickup(R) + Pickup(U)
Put(O,T) + Put(O,U) + Put(O,V)
Pickup(O) + Pickup(R) + Pickup(U)

PICK and PUT are Inconsistent
Pick requires HandEmpty
Put requires !HandEmpty
Building a Planning Graph

Extend the graph with all applicable actions, noops
Determine exclusive actions, propositions

Planning Graph Heuristics

Key idea: Construct an approximation of the reachability graph in polynomial space
- The planning graph computes the possibly reachable states although they aren’t necessarily feasible

What kinds of heuristics could we apply to the planning graph?
Is the solution GraphPlan finds admissible?
How can we program GraphPlan?

Literals: Each thing in our model
\[ i = \text{Instance("name",TYPE)} \]

Variables: Can take on any TYPE thing
\[ v_a = \text{Variable("v_name",TYPE)} \]

Propositions: Relationships
- Proposition("relation", v_a)
- Proposition("relation", i)
- Proposition("relation2", v, i)
- Proposition("relation2", v_a, v_b)
Example

A = Instance("blockA", BLOCK)
B = Instance("blockB", BLOCK)
C = Instance("blockC", BLOCK)
Proposition("on", B, A)
Proposition("on", B, C)
Proposition("on-table", A)
Proposition("on-table", C)
Proposition("clear", B)
Proposition("clear", C)
Proposition("handempty", True)

Piazza Poll – Select All

A = Instance("blockA", BLOCK)
B = Instance("blockB", BLOCK)
C = Instance("blockC", BLOCK)
a. Proposition("on", B, A)
b. Proposition("on", B, C)
c. Proposition("on-table", A)
d. Proposition("on-table", C)
e. Proposition("clear", B)
f. Proposition("clear", C)
How can we program operators?

Operators: the actions we take change state

\[
o = \text{Operator("name",}
\[\text{[]}, \#\text{preconditions}
\[\text{[]}, \#\text{add effects}
\[\]}) \# \text{delete effects}
\]

How can we program operators?

Operators: the actions we take change state

\[
pickup_table = \text{Operator("pick_table",}
\[
\text{[Proposition("Handempty",True),}
\text{Proposition("clear", v_block),}
\text{Proposition("on-table", v_block)],}
\[
\text{[Proposition("holding", v_block)],}
\text{[Proposition("Handempty",True),}
\text{Proposition("on-table", v_block]}
\]
\)
How can we program operators?

Operators: the actions we take change state

```python
pickup_table = Operator("pick_table",
[Proposition("Handempty",True),
 Proposition("clear", v_block),
 Proposition("on-table", v_block)],
[Proposition("holding", v_block)],
[Proposition("Handempty",True),
 Proposition("on-table", v_block])
)
```

How can we program operators?

Operators: the actions we take change state

```python
pickup_table = Operator("pick_table",
[Proposition("Handempty",True),
 Proposition("clear", v_block),
 Proposition("on-table", v_block)],
[Proposition("holding", v_block)],
[Proposition("Handempty",True),
 Proposition("on-table", v_block])
)
```
How can we program operators?

Operators: the actions we take change state

```python
pickup_table = Operator("pick_table",
[Proposition("Handempty",True),
 Proposition("clear", v_block),
 Proposition("on-table", v_block2)],
[Proposition("holding", v_block)],
[Proposition("Handempty",True),
 Proposition("on-table", v_block])
)
```

Variable with matching name must match subsequent propositions

Variables that don’t match name don’t have to be the same

We will give you a GraphPlan Solver

Solver takes all instances, all operators, the start state and the goal state and produces a plan
Another Example - Rocket Ship

Suppose we have a rocket ship that can only be used once. It has to carry two payloads.

Literals?
Another Example - Rocket Ship

Suppose we have a rocket ship that can only be used once. It has to carry two payloads.


Start state:

At(Rocket, LocA), Has-Fuel(), Unloaded(G,LocA), Unloaded(O,LocA)
Another Example - Rocket Ship

Suppose we have a rocket ship that can only be used once. It has to carry two payloads.

Start state:
   At(Rocket, LocA), Has-Fuel(),
   Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
   At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)

Move: Load: Unload:
Another Example - Rocket Ship


Start state:
- At(Rocket, LocA), Has-Fuel(),
- Unloaded(G,LocA), Unloaded(O,LocA)

Goal state:
- At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)

Move:
P: At(Rocket,L)
A: 
D: 

Another Example - Rocket Ship


Start state:
- At(Rocket, LocA), Has-Fuel(),
- Unloaded(G,LocA), Unloaded(O,LocA)

Goal state:
- At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)

Move:
P: At(Rocket,L)
A: 
D: 
Another Example - Rocket Ship


Start state:
- At(Rocket, LocA), Has-Fuel(),
- Unloaded(G,LocA), Unloaded(O,LocA)

Goal state:
- At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)

Move:
P: At(Rocket,L)
A:
D:
Another Example - Rocket Ship

Start state:
   At(Rocket, LocA), Has-Fuel(),
   Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
   At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L

Move:
P: At(Rocket,L), Has-Fuel()
A:
D:

Another Example - Rocket Ship

Start state:
   At(Rocket, LocA), Has-Fuel(),
   Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
   At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L

Move:
P: At(Rocket,L), Has-Fuel()
A: At(Rocket,Dest)
D:
Another Example - Rocket Ship

Start state:  
  At(Rocket, LocA), Has-Fuel(),  
  Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:  
  At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L, Dest

Move:
P: At(Rocket,L), Has-Fuel()  
A: At(Rocket,Dest)
D:

Another Example - Rocket Ship

Start state:  
  At(Rocket, LocA), Has-Fuel(),  
  Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:  
  At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L, Dest

Move:
P: At(Rocket,L), Has-Fuel(), L!=Dest  
A: At(Rocket,Dest)
D:
Another Example - Rocket Ship

Start state:
   At(Rocket, LocA), Has-Fuel(),
   Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
   At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L, Dest

Move:
P: At(Rocket,L), Has-Fuel(), L!=Dest
A: At(Rocket, Dest)
D: Has-Fuel(), At(Rocket,L)

Another Example - Rocket Ship

Start state:
   At(Rocket, LocA), Has-Fuel(),
   Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
   At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L, Dest

Load:
P:
A:
D:
Another Example - Rocket Ship

Start state:
   At(Rocket, LocA), Has-Fuel(),
   Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
   At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L, Dest

Load:
P: At(Rocket,L), Unloaded(Pkg,L)
A:
D:
Another Example - Rocket Ship

Start state:
   At(Rocket, LocA), Has-Fuel(),
   Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
   At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L, Dest, Pkg

Load:
P: At(Rocket,L), Unloaded(Pkg,L)
A:
D:
Another Example - Rocket Ship

Start state:
    At(Rocket, LocA), Has-Fuel(),
    Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
    At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L, Dest, Pkg

Unload:
P: At(Rocket,Dest), Loaded(Pkg,Rocket)
A: Unloaded(Pkg,Dest)
D: Loaded(Pkg,Rocket)

Rocket Ship Planning Graph
Rocket Ship Planning Graph

At(Rocket,LocA)
Has-Fuel()
Unloaded(G,LocA)
Unloaded(O,LocA)

Move
Load(G)
At(Rocket,LocB)
Has-Fuel()
Unloaded(G,LocA)
Unloaded(O,LocA)

Loaded(G,Rocket)
Loaded(O,Rocket)

Mutex Actions
Interference:
Move deletes At which is a precondition of Load
Inconsistent:
Move deletes At but noop adds it
Move deletes Has-Fuel but noop adds it

Mutex Propositions:
- At(Rocket,LocB) and At(Rocket,LocA) because Move and noop are mutex actions
- What else?
At time 1: Move can be performed OR both Load actions
At time 2: Possible plans include:
  Load(G), Load(O), Move(LocB)
  Load(G), Move(LocB)
  Load(O), Move(LocB)

At time 3: What will happen?
At time 3: What will happen? Is the planning graph estimate of 3 timesteps admissible? Are there other admissible heuristics that could use the planning graph?

Planning, Thus Far

Goal States
completely specified

Goal Statements
partially specified

Preference models
objective function
How do these different algorithms fit together?

Suppose I have a robot that can take items to different people, deliver messages, etc.

How can I plan the robot’s tasks? Where should it go, and in what order?
How do these different algorithms fit together?

Suppose I have a robot that can take items to different people, deliver messages, etc.

How can I plan the robot’s tasks? Where should it go, and in what order?

Classical Planning

How do these different algorithms fit together?

Suppose I have a robot that can take items to different people, deliver messages, etc.

Once you have the sequence of locations, how does it plan a path to each destination?

BFS
How do these different algorithms fit together?

Suppose I have a robot that can take items to different people, deliver messages, etc.

Once the robot has a list of locations to navigate, how does it get there?

Motor motion control
Rapidly-exploring Random Trees (obstacles)

How do these different algorithms fit together?

Suppose I have a robot that can take items to different people, deliver messages, etc.

Hierarchical Planning:
High-level: classical
Mid-level: BFS
Low-level: motors
Planning, Thus Far

Goal States  
completely specified

Goal Statements  
partially specified

Preference models  
objective function

Probabilistic States and Actions

Increasing Generality

Summary

Informationless states and actions
   Easy to program
   Can take a lot of memory
   Hard to change the problem (e.g., adding a block)

Predicates and Properties
   Potentially avoids the memory issues
   Many alternate definitions
   Need separate actions for different conditions

Other concerns like uncertainty unmodeled
Another Example: Getting Dressed

States?
Actions?
Goal?
Another Example: Robot Navigation

States?
Actions?
Goal?