Outline of Planning

♦ Search vs. planning
◆ STRIPS operators
♦ Partial-order planning
Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*
Standard search algorithms seem to fail miserably:

After-the-fact heuristic/goal test inadequate
Planning systems do the following:
1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

<table>
<thead>
<tr>
<th></th>
<th><strong>Search</strong></th>
<th><strong>Planning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td>Lisp data structures</td>
<td>Logical sentences</td>
</tr>
<tr>
<td>Actions</td>
<td>Lisp code</td>
<td>Preconditions/outcomes</td>
</tr>
<tr>
<td>Goal</td>
<td>Lisp code</td>
<td>Logical sentence (conjunction)</td>
</tr>
<tr>
<td>Plan</td>
<td>Sequence from $S_0$</td>
<td>Constraints on actions</td>
</tr>
</tbody>
</table>
STRIPS operators

Tidily arranged actions descriptions, restricted language

**ACTION:** $Buy(x)$

**PRECONDITION:** $At(p), Sells(p, x)$

**EFFECT:** $Have(x)$

[Note: this abstracts away many important details!]

Restricted language $\Rightarrow$ efficient algorithm

Precondition: conjunction of positive literals

Effect: conjunction of literals

A complete set of STRIPS operators can be translated into a set of successor-state axioms
ACTION: Go(x,y):
PRECOND:
EFFECT:
Shakey Example, con’t.

ACTION: Go(x,y):

   PRECOND: At(Shakey,x) \land \text{ln}(x,r) \land \text{ln}(y,r)

   EFFECT:
ACTION: Go(x,y):
   PRECOND: At(Shakey,x) \land \text{ln}(x,r) \land \text{ln}(y,r)
   EFFECT: At(Shakey,y) \land \neg(At(Shakey,x))
Shakey Example, con’t.

ACTION: Go(x,y):
   PRECOND: At(Shakey,x) \land \text{ln}(x,r) \land \text{ln}(y,r)
   EFFECT: At(Shakey,y) \land \neg(At(Shakey,x))

ACTION: Push(b,x,y):
   PRECOND:
   EFFECT:
ACTION: Go(x,y):
    PRECOND: At(Shakey,x) \land \text{ln}(x,r) \land \text{ln}(y,r)
    EFFECT: At(Shakey,y) \land \neg(At(Shakey,x))

ACTION: Push(b,x,y):
    PRECOND: At(Shakey,x) \land \text{Pushable}(b)
    EFFECT:
ACTION: Go(x,y):
   PRECOND: At(Shakey,x) \land In(x,r) \land In(y,r)
   EFFECT: At(Shakey,y) \land \neg At(Shakey,x)

ACTION: Push(b,x,y):
   PRECOND: At(Shakey,x) \land Pushable(b)
   EFFECT: At(b,y) \land At(Shakey,y) \land \neg At(b,x) \land \neg At(Shakey,x)
ACTION: Go(x,y):
   PRECOND: At(Shakey,x) ∧ In(x,r) ∧ In(y,r)
   EFFECT: At(Shakey,y) ∧ ¬At(Shakey,x)

ACTION: Push(b,x,y):
   PRECOND: At(Shakey,x) ∧ Pushable(b)
   EFFECT: At(b,y) ∧ At(Shakey,y) ∧ ¬At(b,x) ∧ ¬At(Shakey,x)

ACTION: ClimbUp(b):
   PRECOND:
   EFFECT:
ACTION: Go(x,y):
  PRECOND: At(Shakey,x) \land \text{ln}(x,r) \land \text{ln}(y,r)
  EFFECT: At(Shakey,y) \land \neg(At(Shakey,x))

ACTION: Push(b,x,y):
  PRECOND: At(Shakey,x) \land \text{Pushable}(b)
  EFFECT: At(b,y) \land At(Shakey,y) \land \neg At(b,x) \land \neg At(Shakey,x)

ACTION: ClimbUp(b):
  PRECOND: At(Shakey,x) \land At(b,x) \land \text{Climbable}(b)
  EFFECT:
ACTION: Go(x,y):
  PRECOND: At(Shakey,x) \land In(x,r) \land In(y,r)
  EFFECT: At(Shakey,y) \land \neg(At(Shakey,x))

ACTION: Push(b,x,y):
  PRECOND: At(Shakey,x) \land Pushable(b)
  EFFECT: At(b,y) \land At(Shakey,y) \land \neg At(b,x) \land \neg At(Shakey,x)

ACTION: ClimbUp(b):
  PRECOND: At(Shakey,x) \land At(b,x) \land Climbable(b)
  EFFECT: On(Shakey,b) \land \neg On(Shakey,Floor)
Shakey Example, con’t.

ACTION: Go(x,y):
   PRECOND: At(Shakey,x) ∧ ln(x,r) ∧ ln(y,r)
   EFFECT: At(Shakey,y) ∧ ¬(At(Shakey,x))

ACTION: Push(b,x,y):
   PRECOND: At(Shakey,x) ∧ Pushable(b)
   EFFECT: At(b,y) ∧ At(Shakey,y) ∧ ¬At(b,x) ∧ ¬At(Shakey,x)

ACTION: ClimbUp(b):
   PRECOND: At(Shakey,x) ∧ At(b,x) ∧ Climbable(b)
   EFFECT: On(Shakey,b) ∧ ¬On(Shakey,Floor)

ACTION: ClimbDown(b):
   PRECOND:
   EFFECT:
ACTION: Go(x,y):
    PRECOND: At(Shakey,x) \land \text{ln}(x,r) \land \text{ln}(y,r)
    EFFECT: At(Shakey,y) \land \neg(At(Shakey,x))

ACTION: Push(b,x,y):
    PRECOND: At(Shakey,x) \land \text{Pushable}(b)
    EFFECT: At(b,y) \land At(Shakey,y) \land \neg At(b,x) \land \neg At(Shakey,x)

ACTION: ClimbUp(b):
    PRECOND: At(Shakey,x) \land At(b,x) \land \text{Climbable}(b)
    EFFECT: On(Shakey,b) \land \neg On(Shakey,\text{Floor})

ACTION: ClimbDown(b):
    PRECOND: On(Shakey,b)
    EFFECT:
ACTION: Go(x,y):
   PRECOND: At(Shakey,x) \land \text{ln}(x,r) \land \text{ln}(y,r)
   EFFECT: At(Shakey,y) \land \neg(At(Shakey,x))

ACTION: Push(b,x,y):
   PRECOND: At(Shakey,x) \land \text{Pushable}(b)
   EFFECT: At(b,y) \land At(Shakey,y) \land \neg At(b,x) \land \neg At(Shakey,x)

ACTION: ClimbUp(b):
   PRECOND: At(Shakey,x) \land At(b,x) \land \text{Climbable}(b)
   EFFECT: On(Shakey,b) \land \neg On(Shakey,Floor)

ACTION: ClimbDown(b):
   PRECOND: On(Shakey,b)
   EFFECT: On(Shakey,Floor) \land \neg On(Shakey,b)
ACTION: TurnOn(I):

PRECOND:

EFFECT:
Shakey Example, con’t.

ACTION: TurnOn(l):
  PRECOND: On(Shakey,b) \land At(Shakey,x) \land At(l,x)
  EFFECT:
Shakey Example, con’t.

ACTION: TurnOn(l):
    PRECOND: On(Shakey,b) \land At(Shakey,x) \land At(l,x)
    EFFECT: TurnedOn(l)
Shakey Example, con’t.

ACTION: TurnOn(l):
    PRECOND: On(Shakey,b) \land At(Shakey,x) \land At(l,x)
    EFFECT: TurnedOn(l)

ACTION: TurnOff(l):
    PRECOND:
    EFFECT:
ACTION: TurnOn(l):
    PRECOND: On(Shakey,b) \land At(Shakey,x) \land At(l,x)
    EFFECT: TurnedOn(l)

ACTION: TurnOff(l):
    PRECOND: On(Shakey,b) \land At(Shakey,x) \land At(l,x)
    EFFECT:
Shakey Example, con’t.

ACTION: TurnOn(l):
  PRECOND: On(Shakey, b) \& At(Shakey, x) \& At(l, x)
  EFFECT: TurnedOn(l)

ACTION: TurnOff(l):
  PRECOND: On(Shakey, b) \& At(Shakey, x) \& At(l, x)
  EFFECT: \neg TurnedOn(l)
INITIAL STATE:
In(...) Climbable(...) Pushable(...) At(...) TurnedOn(...)
INITIAL STATE:

\[
\begin{align*}
\text{In}(\text{Switch1}, \text{Room1}) \land \text{In}(\text{Door1}, \text{Room1}) \land \text{In}(\text{Door1}, \text{Corridor}) \\
\text{In}(\text{Switch1}, \text{Room2}) \land \text{In}(\text{Door2}, \text{Room2}) \land \text{In}(\text{Door2}, \text{Corridor}) \\
\text{In}(\text{Switch1}, \text{Room3}) \land \text{In}(\text{Door3}, \text{Room3}) \land \text{In}(\text{Door3}, \text{Corridor}) \\
\text{In}(\text{Switch1}, \text{Room4}) \land \text{In}(\text{Door4}, \text{Room4}) \land \text{In}(\text{Door4}, \text{Corridor}) \\
\text{In}(\text{Shakey}, \text{Room3}) \land \text{At}(\text{Shakey}, \text{XG}) \\
\text{In}(\text{Box1}, \text{Room1}) \land \text{In}(\text{Box2}, \text{Room1}) \land \text{In}(\text{Box3}, \text{Room1}) \land \text{In}(\text{Box4}, \text{Room1})
\end{align*}
\]
INITIAL STATE (con’t.):

\[
\text{Climbable(Box1)} \land \text{Climbable(Box2)} \land \text{Climbable(Box3)} \land \text{Climbable(Box4)} \\
\text{Pushable(Box1)} \land \text{Pushable(Box2)} \land \text{Pushable(Box3)} \land \text{Pushable(Box4)} \\
\text{At(Box1, X1)} \land \text{At(Box2, X2)} \land \text{At(Box3, X3)} \land \text{At(Box4, X4)} \\
\text{TurnedOn(Switch1)} \land \text{TurnedOn(Switch4)}
\]
Plan to achieve goal of getting Box2 into Room2:
Plan to achieve goal of getting Box2 into Room2:
- Go(XS, Door3)
- Go(Door3, Door1)
- Go(Door1, X2)
- Push(Box2, X2, Door1)
- Push(Box2, Door1, Door2)
- Push(Box2, Door2, Switch2)
Partially ordered plans

*Partially ordered* collection of steps with

- *Start* step has the initial state description as its effect
- *Finish* step has the goal description as its precondition
- causal links from outcome of one step to precondition of another
- temporal ordering between pairs of steps

Open condition = precondition of a step not yet causally linked

A plan is *complete* iff every precondition is achieved

A precondition is *achieved* iff it is the effect of an earlier step and no possibly intervening step undoes it
Example

Start

At(Home)  Sells(HWS,Drill)  Sells(SM,Milk)  Sells(SM,Ban.)

Have(Milk)  At(Home)  Have(Ban.)  Have(Drill)

Finish
Example

- Start
  - At(Home)
  - Sells(HWS, Drill)
  - Sells(SM, Milk)
  - Sells(SM, Ban.)

- At(HWS)
  - Sells(HWS, Drill)

- Buy(Drill)
  - At(x)

- Go(SM)
  - At(SM)
  - Sells(SM, Milk)

- Buy(Milk)
  - Have(Milk)

- Finish
  - At(Home)
  - Have(Ban.)
  - Have(Drill)
Example

Start

At(Home)

Go(HWS)

At(HWS) Sells(HWS,Drill)

Buy(Drill)

At(HWS)

Go(SM)

At(SM) Sells(SM,Milk)

Buy(Milk)

At(SM) Sells(SM,Ban.)

Buy(Ban.)

Go(Home)

Have(Milk) At(Home) Have(Ban.) Have(Drill)

Finish
Planning process

Operators on partial plans:
- **add a link** from an existing action to an open condition
- **add a step** to fulfill an open condition
- **order** one step wrt another to remove possible conflicts

Gradually move from incomplete/vague plans to complete, correct plans

Backtrack if an open condition is unachievable or if a conflict is unresolvable
A clobberer is a potentially intervening step that destroys the condition achieved by a causal link. E.g., $Go(Home)$ clobbers $At(Supermarket)$:

**Demotion**: put before $Go(Supermarket)$

**Promotion**: put after $Buy(Milk)$
Properties of POP

Nondeterministic algorithm: backtracks at choice points on failure:
  – choice of $S_{add}$ to achieve $S_{need}$
  – choice of demotion or promotion for clobberer
  – selection of $S_{need}$ is irrevocable

POP is sound, complete, and systematic (no repetition)

Extensions for disjunction, universals, negation, conditionals

Can be made efficient with good heuristics derived from problem description

Particularly good for problems with many loosely related subgoals
Example: Blocks world

"Sussman anomaly" problem

Start State

Goal State

$Clear(x) \land On(x,z) \land Clear(y)$

$\neg On(x,z) \land \neg Clear(y)$

$Clear(z) \land On(x,y)$

$Clear(x) \land On(x,z)$

$\neg On(x,z) \land Clear(z) \land On(x,\text{Table})$

$\text{PutOn}(x,y)$

$\text{PutOnTable}(x)$

+ several inequality constraints

Chapter 11 46
Example contd.

\[
\begin{align*}
\text{On(C,A)} & \quad \text{On(A,Table)} & \quad \text{Cl(B)} & \quad \text{On(B,Table)} & \quad \text{Cl(C)} \\
\end{align*}
\]

On(A,B) \quad On(B,C)

\[
\begin{align*}
\text{FINISH} \\
\end{align*}
\]
Example contd.

\begin{itemize}
  \item On(C,A)
  \item On(A,Table)
  \item Cl(B)
  \item On(B,Table)
  \item Cl(C)
  \item PutOn(B,C)
  \item On(A,B)
  \item On(B,C)
\end{itemize}
Example contd.

\[ \text{On}(C, A) \quad \text{On}(A, \text{Table}) \quad \text{Cl}(B) \quad \text{On}(B, \text{Table}) \quad \text{Cl}(C) \]

- **START**
  - \[ \text{On}(C, A) \quad \text{On}(A, \text{Table}) \quad \text{Cl}(B) \quad \text{On}(B, \text{Table}) \quad \text{Cl}(C) \]
  - **PutOn(A, B)**
    - \[ \text{Cl}(A) \quad \text{On}(A, z) \quad \text{Cl}(B) \]
    - **PutOn(B, C)**
      - \[ \text{On}(A, z) \quad \text{Cl}(B) \quad \text{On}(B, z) \quad \text{Cl}(C) \]

**PutOn(A, B) clobbers Cl(B)** => order after **PutOn(B, C)**

**FINISH**

Chapter 11  49
Example contd.

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

PutOnTable(C)

PutOn(A,B)

PutOn(B,C)

Cl(B) On(B,z) Cl(C)

PutOnTable(C)

PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)

PutOn(B,C) clobbers Cl(C) => order after PutOnTable(C)

On(C,z) Cl(C)

On(C,A) Cl(B) On(B,Table) Cl(C)

On(A,B) On(B,C)

FINISH
Heuristics for Planning

Most obvious Heuristic: Number of distinct open preconditions.
  Overestimates: When actions achieve multiple goals
  Underestimates: When negative interactions between plan steps

Better way: Use planning graph for generating better heuristic estimates.
Planning Graphs

Levels: Correspond to time steps in the plan (0 = initial state)

Each level contains literals + actions: those that *could* be true or executed

Number of planning steps in planning graph is a good estimate of how difficult it is to achieve a given literal from initial state

Can be constructed very efficiently

Works only for *propositionalized problems*
Planning Graph – Have Cake

Init(Have(Cake))
Goal(Have(Cake) ∧ Eaten(Cake))
Action(Eat(Cake))
  Precond: Have(Cake)
  Effect: ¬Have(Cake) ∧ Eaten(Cake))
Action(Bake(Cake))
  Precond: ¬Have(Cake)
  Effect: Have(Cake))

Persistence actions
Mutual exclusion (mutex) links
Mutex Links

A mutex relation holds between two actions at a given level if any of the following is true:

◊ **Inconsistent effects**: one action negates another.

◊ **Interference**: one of effects of action is negation of precondition of another action.

◊ **Competing needs**: one of preconditions of action is mutually exclusive with precondition of other.

A mutex relation holds between two literals at a given level if:

◊ One is negation of other.

◊ Each possible pair of actions that could achieve the literals is mutex.
Heuristics from Planning Graphs

Estimate cost of goal literal = level it first appears = Level Cost

Use serial planning graphs to allow only one action at a time.

Cost of conjunction of goals:
- **Max-level**: Maximum level cost of any goal
- **Level sum**: Sum of level costs of goals (note: inadmissible)
- **Set-level**: Level at which all literals appear without mutex

![Planning Graph Diagram]

\[ S_0 \rightarrow A_0 \rightarrow S_1 \rightarrow A_1 \rightarrow S_2 \]

- \( \text{Have(Cake)} \)
- \( \neg \text{Eaten(Cake)} \)
- \( \text{Eat(Cake)} \)
- \( \text{Bake(Cake)} \)
- \( \text{Eaten(Cake)} \)
Heuristics from Planning Graphs

Estimate cost of goal literal = level it first appears = Level Cost

Use serial planning graphs to allow only one action at a time.

Cost of conjunction of goals:
- **Max-level**: Maximum level cost of any goal
- **Level sum**: Sum of level costs of goals (note: inadmissible)
- **Set-level**: Level at which all literals appear without mutex

\[
\begin{align*}
S_0 & \rightarrow A_0 & S_1 & \rightarrow A_1 & S_2 \\
\text{Have(Cake)} & \rightarrow \text{Eat(Cake)} & \text{Have(Cake)} & \rightarrow \text{Bake(Cake)} & \text{Have(Cake)} \\
\neg \text{Eaten(Cake)} & \rightarrow & \neg \text{Have(Cake)} & \rightarrow & \neg \text{Have(Cake)} \\
\end{align*}
\]

Max-level cost?
Heuristics from Planning Graphs

Estimate cost of goal literal = level it first appears = **Level Cost**

Use **serial planning graphs** to allow only one action at a time.

Cost of conjunction of goals:
- ♦ **Max-level**: Maximum level cost of any goal
- ♦ **Level sum**: Sum of level costs of goals (note: inadmissible)
- ♦ **Set-level**: Level at which all literals appear without mutex

Max-level cost? 1
Heuristics from Planning Graphs

Estimate cost of goal literal = level it first appears = Level Cost

Use serial planning graphs to allow only one action at a time.

Cost of conjunction of goals:

- Max-level: Maximum level cost of any goal
- Level sum: Sum of level costs of goals (note: inadmissible)
- Set-level: Level at which all literals appear without mutex

Max-level cost? 1  Level sum cost?
Heuristics from Planning Graphs

Estimate cost of goal literal = level it first appears = Level Cost

Use serial planning graphs to allow only one action at a time.

Cost of conjunction of goals:

- **Max-level**: Maximum level cost of any goal
- **Level sum**: Sum of level costs of goals (note: inadmissible)
- **Set-level**: Level at which all literals appear without mutex

\[
\begin{array}{c}
S_0 \\
\text{Have(Cake)} \\
\neg\text{Eaten(Cake)}
\end{array}
\quad
\begin{array}{c}
A_0 \\
\text{Eat(Cake)}
\end{array}
\quad
\begin{array}{c}
S_1 \\
\text{Have(Cake)} \\
\neg\text{Have(Cake)} \\
\text{Eaten(Cake)} \\
\neg\text{Eaten(Cake)}
\end{array}
\quad
\begin{array}{c}
A_1 \\
\text{Bake(Cake)} \\
\neg\text{Have(Cake)} \\
\text{Eat(Cake)} \\
\neg\text{Eaten(Cake)}
\end{array}
\quad
\begin{array}{c}
S_2 \\
\text{Have(Cake)} \\
\neg\text{Have(Cake)} \\
\text{Eaten(Cake)} \\
\neg\text{Eaten(Cake)}
\end{array}
\]

Max-level cost? 1  Level sum cost? 1
Heuristics from Planning Graphs

Estimate cost of goal literal = level it first appears = **Level Cost**

Use **serial planning graphs** to allow only one action at a time.

Cost of conjunction of goals:
- **Max-level**: Maximum level cost of any goal
- **Level sum**: Sum of level costs of goals (note: inadmissible)
- **Set-level**: Level at which all literals appear without mutex

```
<table>
<thead>
<tr>
<th>S_0</th>
<th>A_0</th>
<th>S_1</th>
<th>A_1</th>
<th>S_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Have(Cake)</td>
<td></td>
<td>Have(Cake)</td>
<td>Bake(Cake)</td>
<td>Have(Cake)</td>
</tr>
<tr>
<td>Eat(Cake)</td>
<td></td>
<td>\neg Have(Cake)</td>
<td>Eat(Cake)</td>
<td>\neg Eaten(Cake)</td>
</tr>
<tr>
<td>\neg Eaten(Cake)</td>
<td></td>
<td>Eaten(Cake)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Max-level cost? 1     Level sum cost? 1     Set-level Cost?
Heuristics from Planning Graphs

Estimate cost of goal literal = level it first appears = **Level Cost**

Use **serial planning graphs** to allow only one action at a time.

**Cost of conjunction of goals:**

- **Max-level:** Maximum level cost of any goal
- **Level sum:** Sum of level costs of goals (note: inadmissible)
- **Set-level:** Level at which all literals appear without mutex

```
S₀
\underline{\text{Have}(\text{Cake})}
\underline{\text{\neg Eaten}(\text{Cake})}

A₀
\underline{\text{Eat}(\text{Cake})}

S₁
\underline{\text{Have}(\text{Cake})}
\underline{\text{\neg Have}(\text{Cake})}
\underline{\text{\neg Eaten}(\text{Cake})}

A₁
\underline{\text{Bake}(\text{Cake})}
\underline{\text{\neg Have}(\text{Cake})}
\underline{\text{\neg Eaten}(\text{Cake})}

S₂
\underline{\text{Have}(\text{Cake})}
\underline{\text{\neg Have}(\text{Cake})}
\underline{\text{\neg Eaten}(\text{Cake})}
```

Max-level cost? 1  Level sum cost? 1  Set-level Cost? 2
GraphPlan algorithm

Extracting a plan from planning graph...

```
function GRAPHPLAN(problem) returns solution or failure
    graph ← Initial-Planning-Graph(problem)
    goals ← Goals[problem]
    loop do
        if goals all non-mutex in last level of graph, then do
            solution ← Extract-Solution(graph, goals, Length(graph))
            if solution ≠ failure then return solution
            else if No-Solution-Possible(graph) then return failure
        graph ← Expand-Graph(graph, problem)
```
Spare Tire Problem

\textit{Init(At(Flat,Axle) \land At(Spare,Trunk))}
\textit{Goal(At(Spare,Axle))}
\textit{Action(Remove(Spare,Trunk),}
\text{Precond: \textit{At(Spare,Trunk)}}
\text{Effect: \neg\textit{At(Spare,Trunk)} \land \textit{At(Spare,Ground)}}
\textit{Action(Remove(Flat,Axle),}
\text{Precond: \textit{At(Flat,Axle)}}
\text{Effect: \neg\textit{At(Flat,Axle)} \land \textit{At(Flat,Ground)}}
\textit{Action(PutOn(Spare,Axle),}
\text{Precond: \textit{At(Spare,Ground)} \land \neg\textit{At(Flat,Axle)}}
\text{Effect: \neg\textit{At(Spare,Ground)} \land \textit{At(Spare,Axle)}}
\textit{Action(LeaveOvernight,}
\text{Precond:}
\text{Effect: \neg\textit{At(Spare,Ground)} \land \neg\textit{At(Spare,Axle)} \land \neg\textit{At(Spare,Trunk)}
\land \neg\textit{At(Flat,Ground)} \land \neg\textit{At(Flat,Axle)})}
Planning Graph – Spare Tire

(Not all mutex’s shown.)
Example of Inconsistent Effects?
Example of Inconsistent Effects? Remove(Spare, Trunk) and LeaveOvernight
Example of Inconsistent Effects? Remove(Spare, Trunk) and LeaveOvernight
Example of Interference?
Example of Inconsistent Effects? Remove(Spare, Trunk) and LeaveOvernight
Example of Interference? Remove(Flat, Axle) LeaveOvernight
Example of Inconsistent Effects?  Remove(Spare, Trunk) and LeaveOvernight
Example of Interference?  Remove(Flat, Axle) LeaveOvernight
Example of Competing Needs?
Example of Inconsistent Effects?  Remove(Spare, Trunk) and LeaveOvernight
Example of Interference?  Remove(Flat, Axle) and LeaveOvernight
Example of Competing Needs?  PutOn(Spare, Axle) and Remove(Flat, Axle)
Example of Inconsistent Effects? Remove(Spare, Trunk) and LeaveOvernight
Example of Interference? Remove(Flat, Axle) and LeaveOvernight
Example of Competing Needs? PutOn(Spare, Axle) and Remove(Flat, Axle)
Example of Inconsistent Support?
Example of Inconsistent Effects?  Remove(Spare, Trunk) and LeaveOvernight
Example of Interference?  Remove(Flat, Axle) and LeaveOvernight
Example of Competing Needs?  PutOn(Spare, Axle) and Remove(Flat, Axle)
Example of Inconsistent Support?  At(Spare, Axle) and At(Flat, Axle)
Summary of Planning Graphs

◊ Yield useful heuristics of state-space and partial order planners

◊ Consists of multiple layers of literals and actions that can occur at each time step

◊ Includes mutex relations to exclude co-occurrences

◊ Plan can be extracted directly from graph
Planning systems operate on explicit representations of states and actions.

STRIPS language describes actions in terms of preconditions and effects.

Partial-order planning (POP) algorithms explore space of plans without committing to a totally ordered sequence of actions.

POP algorithms work backwards from goal, and are particularly effective on problems amenable to divide-and-conquer.

No consensus on any specific planning approach being the best.