Planning

Chapter 11
First, some examples from Logic

Suppose a knowledge base contains just one sentence: \( \exists x \text{AsHighAs}(x, \text{Everest}) \).

Which of the following are legitimate results of applying Existential Instantiation?

\( \diamond \quad \text{AsHighAs}(\text{Everest, Everest}) \)

\( \diamond \quad \text{AsHighAs}(\text{Kilimanjaro, Everest}) \)

\( \diamond \quad \text{AsHighAs}(\text{Kilimanjaro, Everest}) \land \text{AsHighAs}(\text{BenNevis, Everest}) \)

(after 2 applications)
First, some examples from Logic

Suppose a knowledge base contains just one sentence: $\exists x \text{AsHighAs}(x, \text{Everest})$.

Which of the following are legitimate results of applying Existential Instantiation?

$\Diamond \text{AsHighAs}(\text{Everest}, \text{Everest})$

$\Diamond \text{AsHighAs}(\text{Kilimanjaro}, \text{Everest})$

$\Diamond \text{AsHighAs}(\text{Kilimanjaro}, \text{Everest}) \land \text{AsHighAs}(\text{BenNevis}, \text{Everest})$

(after 2 applications)

Numbers 2 and 3 are correct.

Why isn’t number 1 correct?

Does number 3 mean there are now 2 mountains as high as \textit{Everest}?
Another Logic example

“Brothers and sisters have I none, but that man’s father is my father’s son”. Who is that man?
Another Logic example

“Brothers and sisters have I none, but that man’s father is my father’s son”. Who is that man?

Let $\text{Rel}(r, x, y)$ say that family relationship $r$ holds between $x$ and $y$.

\[ Me = \text{me, and } MrX = \text{“that man”} . \]

1. $\text{Rel}($Sibling, $Me, x) \Rightarrow False$
2. $\text{Male}(MrX)$
3. $\text{Rel}($Father, $FX, MrX)$
4. $\text{Rel}($Father, $FM, Me)$
5. $\text{Rel}($Son, $FX, FM)$
Want to show that $Me$ is the only son of my father, and therefore that $Me$ is the father of $MrX$, who is male, and therefore that “that man” is my son.

(6) $Rel(Parent, x, y) \land Male(x) \iff Rel(Father, x, y)$
(7) $Rel(Son, x, y) \iff Rel(Parent, y, x) \land Male(x)$
(8) $Rel(Sibling, x, y) \iff x \neq y \land \exists p Rel(Parent, p, x) \land Rel(Parent, p, y)$
(9) $Rel(Father, x_1, y) \land Rel(Father, x_2, y) \implies x_1 = x_2$

Our query: (Q) $Rel(r, MrX, y)$

We want the answer: $\{r/Son, y/Me\}$
Translating 1-9, we get:

(6a) \(\text{Rel}(\text{Parent}, x, y) \land \text{Male}(x) \Rightarrow \text{Rel}(\text{Father}, x, y)\)
(6b) \(\text{Rel}(\text{Father}, x, y) \Rightarrow \text{Male}(x)\)
(6c) \(\text{Rel}(\text{Father}, x, y) \Rightarrow \text{Rel}(\text{Parent}, x, y)\)
(7a) \(\text{Rel}(\text{Son}, x, y) \Rightarrow \text{Rel}(\text{Parent}, y, x)\)
(7b) \(\text{Rel}(\text{Son}, x, y) \Rightarrow \text{Male}(x)\)
(7c) \(\text{Rel}(\text{Parent}, y, x) \land \text{Male}(x) \Rightarrow \text{Rel}(\text{Son}, x, y)\)
(8a) \(\text{Rel}(\text{Sibling}, x, y) \Rightarrow x \neq y\)
(8b) \(\text{Rel}(\text{Sibling}, x, y) \Rightarrow \text{Rel}(\text{Parent}, P(x, y), x)\)
(8c) \(\text{Rel}(\text{Sibling}, x, y) \Rightarrow \text{Rel}(\text{Parent}, P(x, y), y)\)
(8d) \(\text{Rel}(\text{Parent}, P(x, y), x) \land \text{Rel}(\text{Parent}, P(x, y), y) \land x \neq y \Rightarrow \text{Rel}(\text{Sibling}, x, y)\)
(9) \(\text{Rel}(\text{Father}, x_1, y) \land \text{Rel}(\text{Father}, x_2, y) \Rightarrow x_1 = x_2\)
(N) \(\text{True} \Rightarrow x = y \lor x \neq y\)
(N') \(x = y \land x \neq y \Rightarrow \text{False}\)
(Q') \(\text{Rel}(r, MrX, y) \Rightarrow \text{False}\)
Using resolution to prove \( Q' \) is a contradiction, we get the following:

\( (10: \ 4,6c) \)  \( \text{Rel}(Parent, FM, Me) \)
\( (11: \ 5,7a) \)  \( \text{Rel}(Parent, FM, FX) \)
\( (12: \ 10,8d) \)  \( \text{Rel}(Parent, FM, y) \land Me \neq y \Rightarrow \text{Rel}(Sibling, Me, y) \)
\( (13: \ 12,1) \)  \( \text{Rel}(Parent, FM, y) \land Me \neq y \Rightarrow \text{False} \)
\( (14: \ 13,11) \)  \( Me \neq FX \Rightarrow \text{False} \)
\( (15: \ 14,N) \)  \( Me = FX \)
\( (16: \ 15,3) \)  \( \text{Rel}(Father, Me, MrX) \)
\( (17: \ 16,6c) \)  \( \text{Rel}(Parent, Me, MrX) \)
\( (18: \ 17,2,7c) \)  \( \text{Rel}(Son, MrX, Me) \)
\( (19: \ 18,Q') \)  \( \text{False}\{r/Son, y/Me\} \)
Outline of Planning

- Search vs. planning
- STRIPS operators
- Partial-order 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>Search</th>
<th>Planning</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:
ACTION: Go(x,y):
    PRECOND: At(Shakey,x) \land ln(x,r) \land 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 (\text{At}(\text{Shakey},x))
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:
   EFFECT:
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:
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: 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: \(\text{At}(\text{Shakey},x) \land \text{In}(x,r) \land \text{In}(y,r)\)
   EFFECT: \(\text{At}(\text{Shakey},y) \land \neg(\text{At}(\text{Shakey},x))\)

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

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

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

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

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

ACTION: ClimbDown\((b)\):
  \text{PRECOND: } On(Shakey,b)
  \text{EFFECT: }
Shakey Example, con’t.

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

ACTION: ClimbDown(b):
    PRECOND: On(Shakey,b)
    EFFECT: On(Shakey,Floor) ∧ ¬On(Shakey,b)
ACTION: TurnOn(l):
   PRECOND:
   EFFECT:
ACTION: TurnOn(l):
   PRECOND: On(Shakey,b) \land At(Shakey,x) \land At(l,x)
   EFFECT:
ACTION: TurnOn(l):
  PRECOND: On(Shakey,b) \land At(Shakey,x) \land At(l,x)
  EFFECT: TurnedOn(l)
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:
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: ¬TurnedOn(l)
INITIAL STATE:
In(...)  Climbable(...)  Pushable(...)  At(...)  TurnedOn(...)
INITIAL STATE:

\[
\begin{aligned}
&\text{In}(\text{Switch1},\text{Room1}) \land \text{In}(\text{Door1},\text{Room1}) \land \text{In}(\text{Door1},\text{Corridor}) \\
&\text{In}(\text{Switch1},\text{Room2}) \land \text{In}(\text{Door2},\text{Room2}) \land \text{In}(\text{Door2},\text{Corridor}) \\
&\text{In}(\text{Switch1},\text{Room3}) \land \text{In}(\text{Door3},\text{Room3}) \land \text{In}(\text{Door3},\text{Corridor}) \\
&\text{In}(\text{Switch1},\text{Room4}) \land \text{In}(\text{Door4},\text{Room4}) \land \text{In}(\text{Door4},\text{Corridor}) \\
&\text{In}(\text{Shakey},\text{Room3}) \land \text{At}(\text{Shakey},\text{XS}) \\
&\text{In}(\text{Box1},\text{Room1}) \land \text{In}(\text{Box2},\text{Room1}) \land \text{In}(\text{Box3},\text{Room1}) \land \text{In}(\text{Box4},\text{Room1})
\end{aligned}
\]
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.)

Buy(Drill)

At(HWS)  Sells(HWS, Drill)

Buy(Milk)

At(SM)  Sells(SM, Milk)

Go(SM)

At(x)

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

Finish
Example

At(SM)  At(Home)  Go(HWS)  Buy(Drill)  Go(HWS)  Sells(HWS,Drill)  Buy(Drill)  Go(SM)  Sells(SM,Milk)  Sells(SM,Ban.)  Go(Home)  Have(Milk)  At(Home)  Have(Ban.)  Have(Drill)  Finish

Start

Go(HWS)  Sells(HWS,Drill)

Go(SM)  Sells(SM,Milk)  Sells(SM,Ban.)

Go(Home)
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
function POP(initial, goal, operators) returns plan

    plan ← MAKE-MINIMAL-PLAN(initial, goal)
    loop do
        if SOLUTION?(plan) then return plan
        S_{need}, c ← SELECT-SUBGOAL(plan)
        CHOOSE-OPERATOR(plan, operators, S_{need}, c)
        RESOLVE-THREATS(plan)
    end

function SELECT-SUBGOAL(plan) returns S_{need}, c

    pick a plan step S_{need} from STEPS(plan)
    with a precondition c that has not been achieved
    return S_{need}, c
**POP algorithm contd.**

**procedure** `CHOOSE-OPERATOR(plan, operators, S\text{need}, c)`

- choose a step $S_{add}$ from `operators` or `STEPS(plan)` that has $c$ as an effect
- if there is no such step then fail
- add the causal link $S_{add} \xrightarrow{c} S_{\text{need}}$ to `LINKS(plan)`
- add the ordering constraint $S_{add} \prec S_{\text{need}}$ to `ORDERINGS(plan)`
- if $S_{add}$ is a newly added step from `operators` then
  - add $S_{add}$ to `STEPS(plan)`
  - add `Start \prec S_{add} \prec Finish` to `ORDERINGS(plan)`

**procedure** `RESOLVE-THREATS(plan)`

- for each $S_{\text{threat}}$ that threatens a link $S_i \xrightarrow{c} S_j$ in `LINKS(plan)` do
  - choose either
    - *Demotion:* Add $S_{\text{threat}} \prec S_i$ to `ORDERINGS(plan)`
    - *Promotion:* Add $S_j \prec S_{\text{threat}}$ to `ORDERINGS(plan)`
  - if not `CONSISTENT(plan)` then fail

end
Clobbering and promotion/demotion

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

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
</table>

Goal State

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
</table>

\[
\text{Clear}(x) \land \text{On}(x,z) \land \text{Clear}(y) \\
\text{PutOn}(x,y) \\
\neg \text{On}(x,z) \land \neg \text{Clear}(y) \\
\text{Clear}(z) \land \text{On}(x,y)
\]

\[
\text{Clear}(x) \land \text{On}(x,z) \\
\text{PutOnTable}(x) \\
\neg \text{On}(x,z) \land \text{Clear}(z) \land \text{On}(x,\text{Table})
\]

+ several inequality constraints
Example contd.

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

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

\[ \text{START} \]

\[ \text{FINISH} \]
Example contd.

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

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

Cl(A) On(A,z) Cl(B)

PutOn(A,B)

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

PutOn(B,C)

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

FINISH

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

PutOnTable(C)

PutOn(A,B)

Cl(A) On(A,z) Cl(B)

PutOn(B,C)

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

On(C,z) Cl(C)

PutOnTable(C)

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

FINISH

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

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

\[ \text{Max-level cost?} \]
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

```
S_0  A_0  S_1  A_1  S_2
Have(Cake)  Eat(Cake)
\neg Eaten(Cake)
```

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

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

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  A₀  S₁  A₁  S₂

```
Have(Cake)  
\neg Eaten(Cake)

\neg Have(Cake)  
\neg Eaten(Cake)

Eaten(Cake)  
\neg Eaten(Cake)

Bake(Cake)

Have(Cake)  
\neg Have(Cake)

\neg Eaten(Cake)

Eaten(Cake)  
\neg Eaten(Cake)
```

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

Extracting a plan from planning graph...

\[
\text{function } \text{GraphPlan}(\text{problem}) \text{ returns solution or failure}
\]
\[
\text{graph} \leftarrow \text{Initial-Planning-Graph}(\text{problem})
\]
\[
\text{goals} \leftarrow \text{Goals}[\text{problem}]
\]
\[
\text{loop do}
\]
\[
\text{if goals all non-mutex in last level of graph, then do}
\]
\[
\text{solution} \leftarrow \text{Extract-Solution}(\text{graph}, \text{goals, Length(\text{graph}))}
\]
\[
\text{if solution} \neq \text{failure then return solution}
\]
\[
\text{else if No-Solution-Possible(\text{graph}) then return failure}
\]
\[
\text{graph} \leftarrow \text{Expand-Graph}(\text{graph, problem})
\]
Spare Tire Problem

\[\text{Init}(\text{At}(\text{Flat}, \text{Axle}) \land \text{At}(\text{Spare}, \text{Trunk}))\]

\[\text{Goal}(\text{At}(\text{Spare}, \text{Axle}))\]

\[\text{Action}(\text{Remove}(\text{Spare}, \text{Trunk}), \]
\hspace{2em} \text{Precond: } \text{At}(\text{Spare}, \text{Trunk})
\hspace{2em} \text{Effect: } \neg \text{At}(\text{Spare}, \text{Trunk}) \land \text{At}(\text{Spare}, \text{Ground}))\]

\[\text{Action}(\text{Remove}(\text{Flat}, \text{Axle}), \]
\hspace{2em} \text{Precond: } \text{At}(\text{Flat}, \text{Axle})
\hspace{2em} \text{Effect: } \neg \text{At}(\text{Flat}, \text{Axle}) \land \text{At}(\text{Flat}, \text{Ground}))\]

\[\text{Action}(\text{PutOn}(\text{Spare}, \text{Axle}), \]
\hspace{2em} \text{Precond: } \text{At}(\text{Spare}, \text{Ground}) \land \neg \text{At}(\text{Flat}, \text{Axle})
\hspace{2em} \text{Effect: } \neg \text{At}(\text{Spare}, \text{Ground}) \land \text{At}(\text{Spare}, \text{Axle}))\]

\[\text{Action}(\text{LeaveOvernight}, \]
\hspace{2em} \text{Precond:} \]
\hspace{4em} \text{Effect: } \neg \text{At}(\text{Spare}, \text{Ground}) \land \neg \text{At}(\text{Spare}, \text{Axle}) \land \neg \text{At}(\text{Spare}, \text{Trunk})
\hspace{4em} \land \neg \text{At}(\text{Flat}, \text{Ground}) \land \neg \text{At}(\text{Flat}, \text{Axle}))\]
Planning Graph – Spare Tire

(Not all mutex’s shown.)

S

0

A

0

S

1

A

1

S

2

At(Spare,Trunk)

At(Spare,Trunk)

At(Spare,Trunk)

PutOn(Spare,Axle)

Remove(Spare,Trunk)

Remove(Flat,Axle)

LeaveOvernight

Remove(Flat,Ground)

Remove(Spare,Ground)

Remove(Flat,Ground)

Remove(Spare,Axle)

PutOn(Spare,Axle)

At(Flat,Ground)

At(Flat,Ground)

At(Flat,Axle)

At(Flat,Axle)

At(Spare,Axle)

At(Spare,Axle)

At(Flat,Ground)

At(Flat,Ground)

At(Spare,Ground)

At(Spare,Ground)
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
Planning Graph – Spare Tire

(Not all mutex’s shown.)

Example of Inconsistent Effects? Remove(Spare, Trunk) and LeaveOvernight
Example of Interference? Remove(Flat, Axle) LeaveOvernight
Example of Competing Needs?
Planning Graph – Spare Tire

(Not all mutex’s shown.)

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
Summary

♦ 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.