CS 4649/7649 RIP
Robot Intelligence: Planning

Approaches to Classical Planning

Sungmoon Joo

School of Interactive Computing
College of Computing
Georgia Institute of Technology

*Slides based on Dr. Mike Stilman's lecture slides

Course Info.

- **Course Website: joosm.github.io/RIP2014**
  - **Lecture slides**
  - **Project examples**
  - **Wiki: https://github.com/joosm/RIP2014Wiki/wiki**
- **Email me(sungmoon.joo@cc.gatech.edu)**
  - **Introduce yourself, your github id**
  - **Project ideas**

STRIPS: Blocks World Example

Constants:

$A, B, C, Table$
$IsBlock(A), IsBlock(B)\ldots$

Ground Literals:

$On(B, A)$ $On(C, Table)$
$Clear(B)$ $Clear(Table)$

Actions: $move(x, y, z)$

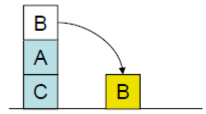
PC: $On(x, y)$, $Clear(x)$, $Clear(z)$

D: $On(x, y)$, $Clear(z)$

A: $On(x, z)$, $Clear(y)$, $Clear(Table)$

Domain Axioms:

$$On(y, x) \wedge On(z, x) \wedge (x \neq Table) \Rightarrow (y = z)$$



$move(B, A, Table)$

Delete: S0	Add: S1
$On(B, A)$, $Clear(Table)$	$On(B, Table)$, $Clear(A)$, $Clear(Table)$
Unchanged	
$On(A, C)$, $On(C, Table)$, $Clear(B)$	$On(A, C)$, $On(C, Table)$, $Clear(B)$

STRIPS Planner*

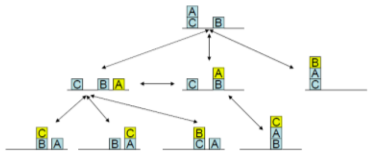
STRIPS(*init*,*goals*)
1. *state* = *init*, *plan* = [], *stack* = []
2. Push *goals* on *stack*
3. Repeat until *stack* = []
   - If *top* = *state* Pop *stack*
   - If *top* = conjunctive then Push subgoals onto *stack*.
   - If *top* = simple literal
     - Choose operator O where A_O = *top*
     - Replace *top* with operator O
     - Push preconditions of O onto *stack*
   - If top = operator O then
     - *state* = apply(O,*state*)
     - *plan* = [*plan*, O]

***STRIPS (Stanford Research Institute Problem Solver)**

Means – Ends Analysis

- Means-Ends Analysis

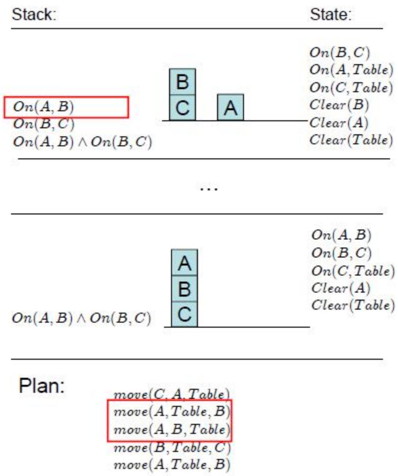
- Search only relevant aspect of problem
- What means (operators) are available to achieve the desired ends(goal)
- Find difference between goal and current state
- Find operator to reduce the difference
- Perform means-ends analysis on new subgoals

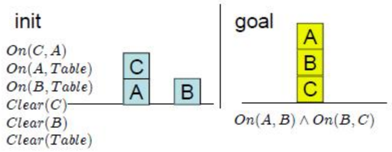


Properties of a Planner

1) Sound: The planner produces valid plans
   - STRIPS is sound
2) Optimal: The planner produces optimal (shortest) plans
   - STRIPS is suboptimal
3) Complete: The planner finds a solution when there is one or returns that the solution is not possible.
   - STRIPS is incomplete!

Sussman Anomaly





Gerald Sussman

(MIT since 1964)

*Called 'anomaly' because it seemed to make sense for a conjunctive goals to first achieve one goal and then achieve another goal, and then the complete goal would be achieved.

Airplane Logistics

Load(o,p,loc)
PC: At(o,loc)
At(p,loc)
Add: In(o,p)
Delete: At(o,loc)

Unload(o,p,loc)
PC: In(o,p)
At(p,loc)
Add: At(o,loc)
Delete: In(o,p)

Fly(p,from,to)
PC: At(p,from)
Have-Fuel(p)
Add: At(p,to)
Delete: At(p,from)
Have-Fuel(p)

Init: At(O_1, A) At(O_2,A) Plane(747)
At(747, A) Have-Fuel(747)

Goal: At(O_1, B) $\wedge$ At(O_2,B)

Airplane Logistics

Stack:	State:
$At(O_1, B)$ $At(O_2, B)$ $At(O_1, B) \wedge At(O_2, B)$	$At(O_1, A)$ $At(O_2, A)$ $At(747, A)$ $HaveFuel(747)$
$\boxed{In(O_1, 747)}$ $At(747, B)$ $Unload(O_1, 747, B)$ $At(O_2, B)$ $At(O_1, B) \wedge At(O_2, B)$	$At(O_1, A)$ $At(O_2, A)$ $At(747, A)$ $HaveFuel(747)$
$\cancel{At(O_1, A)}$ $\cancel{At(747, A)}$ $\boxed{Load(O_1, 747, A)}$ $At(747, B)$ $Unload(O_1, 747, B)$ $At(O_2, B)$ $At(O_1, B) \wedge At(O_2, B)$	$At(O_1, A)$ $At(O_2, A)$ $At(747, A)$ $HaveFuel(747)$

Airplane Logistics

Stack:	State:
$At(O_1, B)$ $At(O_2, B)$ $At(O_1, B) \wedge At(O_2, B)$	$At(O_1, A)$ $At(O_2, A)$ $At(747, A)$ $HaveFuel(747)$
$In(O_1, 747)$ $At(747, B)$ $Unload(O_1, 747, B)$ $At(O_2, B)$ $At(O_1, B) \wedge At(O_2, B)$	$At(O_1, A)$ $At(O_2, A)$ $At(747, A)$ $HaveFuel(747)$
$At(O_1, A)$ $At(747, A)$ $Load(O_1, 747, A)$ $At(747, B)$ $Unload(O_1, 747, B)$ $At(O_2, B)$ $At(O_1, B) \wedge At(O_2, B)$	$At(O_1, A)$ $At(O_2, A)$ $At(747, A)$ $HaveFuel(747)$
$At(747, B)$ $Unload(O_1, 747, B)$ $At(O_2, B)$ $At(O_1, B) \wedge At(O_2, B)$	$\boxed{In(O_1, 747)}$ $At(O_2, A)$ $At(747, A)$ $HaveFuel(747)$

Airplane Logistics

Stack:	State:
$At(747, B)$ $Unload(O_1, 747, B)$ $At(O_2, B)$ $At(O_1, B) \wedge At(O_2, B)$	$In(O_1, 747)$ $At(O_2, A)$ $At(747, A)$ $HaveFuel(747)$
$At(747, A)$ $HaveFuel(747)$ $\boxed{Fly(747, A, B)}$ $Unload(O_1, 747, B)$ $At(O_2, B)$ $At(O_1, B) \wedge At(O_2, B)$	$In(O_1, 747)$ $At(O_2, A)$ $At(747, A)$ $HaveFuel(747)$
$Unload(O_1, 747, B)$ $At(O_2, B)$ $At(O_1, B) \wedge At(O_2, B)$	$In(O_1, 747)$ $At(O_2, A)$ $At(747, B)$

Plan: $Load(O_1, 747, A)$
$Fly(747, A, B)$

Airplane Logistics

Stack:	State:
$At(O_2, B)$ $At(O_1, B) \wedge At(O_2, B)$	$At(O_1, B)$ $At(O_2, A)$ $At(747, B)$
$In(O_2, 747)$ $At(747, B)$ $Unload(O_2, 747, B)$ $At(O_2, B)$ $At(O_1, B) \wedge At(O_2, B)$	$At(O_1, B)$ $At(O_2, A)$ $At(747, B)$
$At(O_2, A)$ $\boxed{At(747, A)}$ $Load(O_2, 747, A)$ $At(747, B)$ $Unload(O_2, 747, B)$ $At(O_2, B)$ $At(O_1, B) \wedge At(O_2, B)$	$At(O_1, B)$ $At(O_2, A)$ $At(747, B)$
$At(747, B)$ $HaveGas(747)$ ← No action that adds this! $\boxed{Fly(747, B, A)}$ $Load(O_2, 747, A)$ $At(747, B)$ $Unload(O_2, 747, B)$ $At(O_2, B)$ $At(O_1, B) \wedge At(O_2, B)$	$At(O_1, B)$ $At(O_2, A)$ $At(747, B)$

Plan: $Load(O_1, 747, A)$
$Fly(747, A, B)$
$Unload(O_1, 747, B)$
$\ldots?$

Was there a way to solve this problem?

Load(o,p,loc)
PC: At(o,loc)
At(p,loc)
Add: In(o,p)
Delete: At(o,loc)

Unload(o,p,loc)
PC: In(o,p)
At(p,loc)
Add: At(o,loc)
Delete: In(o,p)

Fly(p,from,to)
PC: At(p,from)
Have-Fuel(p)
Add: At(p,to)
Delete: At(p,from)
Have-Fuel(p)

Init: $At(O_1, A)$ $At(O_2,A)$ Plane(747)
At(747, A) Have-Fuel(747)

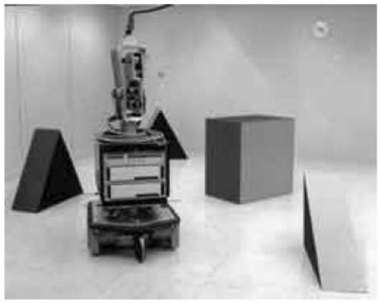
Goal: $At(O_1, B) \wedge At(O_2,B)$

Properties of a Planner

1) Sound: The planner produces valid plans
   - STRIPS is sound
2) Optimal: The planner produces optimal (shortest) plans
   - STRIPS is suboptimal
3) Complete: The planner finds a solution when there is one or returns that the solution is not possible.
   - STRIPS is incomplete!

STRIPS: Shakey SRI 1966-1972

- "SHAKEY: Experimenation in Robot Learning and **Planning**"
- "First Electronic Person" – Life Magazine (1970)
- Cameras, Tactile Sensors
- 1.35 MB of code



http://www.sri.com/work/timeline-innovation/timeline.php?timeline=computing-digital#!&innovation=shakey-the-robot
http://www.ai.sri.com/shakey/

Shakey Unblocks a Door

http://www.sri.com/newsroom/video/shakey-experimentation-robot-learning-and-planning-1969

Shakey: Pushing

PUSH(OB,X,Y)

Preconditions:

(∃RX)[INROOM(ROBOT,RX) ∧
INROOM(OB,RX) ∧ LOCINROOM(X,Y,RX)]
∧ PUSHABLE(OB)

Delete List:

AT(ROBOT,$1,$2)
NEXTTO(ROBOT,$1)
AT(OB,$1,$2)
NEXTTO(OB,$1)
NEXTTO($1,OB)

Add List:

*AT(OB,X,Y)
NEXTTO(ROBOT,OB)

Raphael, B., Duda, R. O., Fikes, R. E., Hart, P. E., Nilsson, N. J., Thorndyke, P. W. and Wilber, B. M. **Research and Applications - Artificial Intelligence**, Technical Report . Stanford Research Institute, December 1971.

Shakey: Unblocking, why not Pushing?

UNBLOCK(DX,RX,BX)

Preconditions:

BLOCKED(DX,RX,BX) ∧ INROOM(ROBOT,RX) ∧ PUSHABLE(BX)

Delete List:

AT(ROBOT,$1,$2)
BLOCKED(DX,RX,BX)
AT(BX,$1,$2)
NEXTTO(ROBOT,$1)
NEXTTO(BX,$1)
NEXTTO($1,BX)

Add List:

*UNBLOCKED(DX,RX)
NEXTTO(ROBOT,BX)

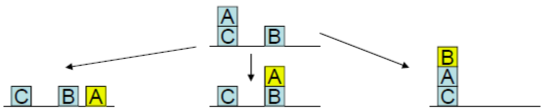
Raphael, B., Duda, R. O., Fikes, R. E., Hart, P. E., Nilsson, N. J., Thorndyke, P. W. and Wilber, B. M. **Research and Applications - Artificial Intelligence**, Technical Report . Stanford Research Institute, December 1971.

Linear Planning Analysis

- Stack search was greedy to achieve sub-goals
- "Depth First" with incomplete backtracking
- Not effective when sub-goals interact
- But we like Means-Ends because it is efficient
- Is there a better way?

Observations

- Forward search was undesirable due to options in initial state

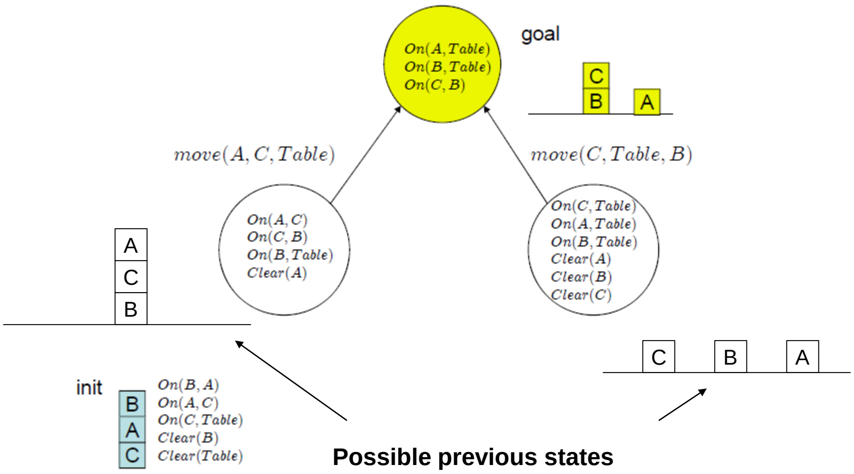


- Goal state has fewer conjuncts than ground literals in initial state

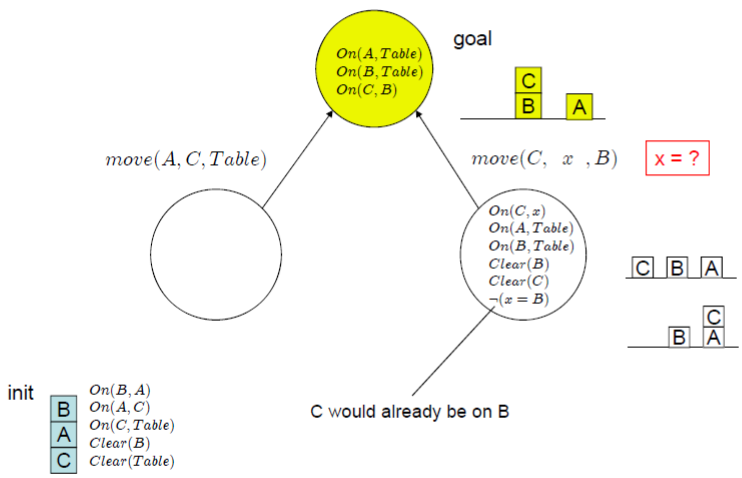
Goal: $On(A, Table)$ $On(B, Table)$ $On(C, B)$ **<** Init: $On(B, A)$ $On(A, C)$ $On(C, Table)$ $Clear(B)$ $Clear(Table)$

- Lets search backwards!

Regression Planning



Regression Planning (Variables Show Up)



Regression Planning

- Just as with forward (progression) planning this algorithm would be complete. However:
- Still large branching factor (potentially larger)
- When do we instantiate the variables?

Regression Planning

- Still large branching factor (potentially larger)
- When do we instantiate the variables?
- Introduces new concept:

"Least Commitment Planning"

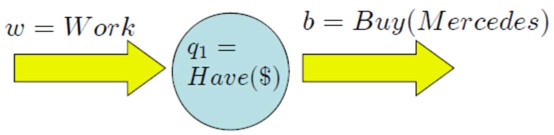
Make choices **only** when they are relevant to solving
The current part of the problem

Partial Order Planning

- Based on the concept of "Least Commitment"
- Nodes are partial plans
- Arcs/Transitions are **plan refinements**
- Solution is a node, not a path (search in plan space!!)

Partial Order Planning

- A plan consists of:
  - **A: Set of actions**
  - **O: Set of orderings for actions (a < b)**
  - **Q: Set of causal links**
- A causal link $q_1 \in Q$ is defined as follows:
  - Action b has a precondition that is established by Action w



Partial Order Planning: Threats

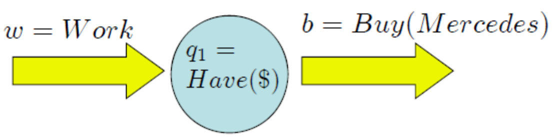
- A threat to (a,q,b) is defined as an action **t** that:
  - Has ¬ q as an effect

    $\neg q \in t_{add}$

  - Could occur between *a* and *b*

    $O \cup \; a < t < b$ is inconsistent

- What action **t** would be a threat to causal link q_1?



Partial Order Planning: Initialization

Since we're only talking about actions, lets turn states into actions:

- A_0
  - **No preconditions**
  - Initial state as **effects**
  - Must be the **first step** in the plan (all actions > A_0 in O)

- A_N
  - **No effects**
  - Goals as **preconditions**
  - Must be the **last step** in the plan (all actions < A_N in O)

POP Algorithm: Simplified UCPOP* (Weld)

POP ((A,O,Q), agenda, actions)
If agenda=∅ then return(A,O,Q)
(q, a_{need}) = **Choose**(pair) from agenda
a_{add} = **Choose**(actions) s.t. $q \in Add(a_{add})$
If no such action a_{add} exists, **Fail**

$Q' = Q \cup (a_{add}, q, a_{need})$
$O' = O \cup (a_{add} < a_{need})$
agenda' = agenda - (q, a_{need})

If a_{add} is new, then $A = A \cup a_{add}$ and
$\forall p \in PC(a_{add})$, add (p, a_{add}) to agenda'

For every action a_t that threatens any causal link (a_i, Q, a_j) in Q'
Choose to add $a_t < a_i$ or $a_j < a_t$ to O
if neither choice is consistent, **Fail**

POP((A',O',Q'), agenda, actions)

The magic "Choose" enables backtracking

***UCPOP (Universal, Conditional Partial-Order Planner)**

POP Algorithm

POP ((A,O,Q), agenda, actions)
If agenda=∅ then return(A,O,Q)
(q, a_{need}) = **Choose**(pair) from agenda
a_{add} = **Choose**(action) s.t. $q \in Add(a_{add})$
If no such action a_{add} exists, **Fail**

$Q' = Q \cup (a_{add}, q, a_{need})$
$O' = O \cup (a_{add} < a_{need})$
agenda' = agenda - (q, a_{need})

If a_{add} is new, then $A = A \cup a_{add}$ and
$\forall p \in PC(a_{add})$, add (p, a_{add}) to agenda'

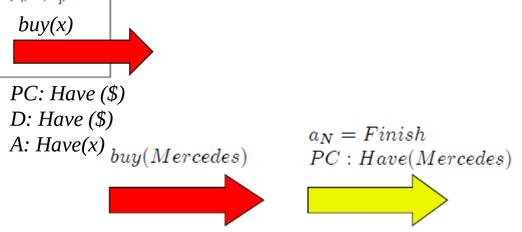
For every action a_t that threatens any causal link (a_i, Q, a_j) in Q'
Choose to add $a_t < a_i$ or $a_j < a_t$ to O
if neither choice is consistent, **Fail**

POP((A',O',Q'), agenda, actions)

Agenda ={ $(Have(\$), buy(Merc.))$
}

Q = {$(buy(Merc.), Have(Merc.), Finish)$
}

O = { $buy(Merc.) < Finish$
}



POP Algorithm

POP ((A,O,Q), agenda, actions)
If agenda=∅ then return(A,O,Q)
(q, a_{need}) = **Choose**(pair) from agenda
a_{add} = **Choose**(action) s.t. $q \in Add(a_{add})$
If no such action a_{add} exists, **Fail**

$Q' = Q \cup (a_{add}, q, a_{need})$
$O' = O \cup (a_{add} < a_{need})$
agenda' = agenda - (q, a_{need})

If a_{add} is new, then $A = A \cup a_{add}$ and
$\forall p \in PC(a_{add})$, add (p, a_{add}) to agenda'

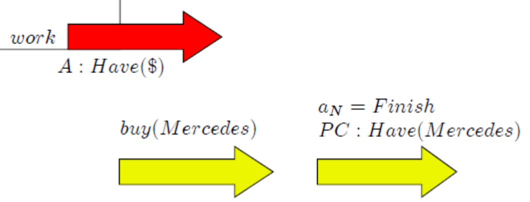
For every action a_t that threatens any causal link (a_i, Q, a_j) in Q'
Choose to add $a_t < a_i$ or $a_j < a_t$ to O
if neither choice is consistent, **Fail**

POP((A',O',Q'), agenda, actions)

Agenda ={
}

Q = {$(buy(Merc.), Have(Merc.), Finish)$
$(work, Have(\$), buy(Merc.))$
}

O = { $buy(Merc.) < Finish$
$work < buy(Merc.)$
}



POP Algorithm

POP ((A,O,Q), agenda, actions)
If agenda=∅ then return(A,O,Q)
(q, a_{need}) = **Choose**(pair) from agenda
a_{add} = **Choose**(action) s.t. $q \in Add(a_{add})$
If no such action a_{add} exists, **Fail**

$Q' = Q \cup (a_{add}, q, a_{need})$
$O' = O \cup (a_{add} < a_{need})$
agenda' = agenda - (q, a_{need})

If a_{add} is new, then $A = A \cup a_{add}$ and
$\forall p \in PC(a_{add})$, add (p, a_{add}) to agenda'

For every action a_t that threatens any causal link (a_i, Q, a_j) in Q'
Choose to add $a_t < a_i$ or $a_j < a_t$ to O
if neither choice is consistent, **Fail**

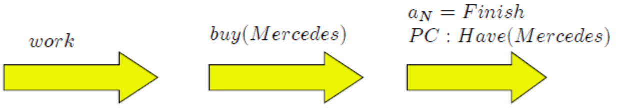
POP((A',O',Q'), agenda, actions)

Agenda ={
}

Q = {$(buy(Merc.), Have(Merc.), Finish)$
$(work, Have(\$), buy(Merc.))$
}

O = { $buy(Merc.) < Finish$
$work < buy(Merc.)$
}

DONE!



POP Algorithm Details

Protect Causal Links use O:

Demotion:

$a_t < a_i$

Promotion:

$a_t > a_j$

POP ((A,O,Q), agenda, actions)

If agenda=∅ then return(A,O,Q)

(q,a_{need}) = **Choose**(pair) from agenda

a_{add} = **Choose**(actions) s.t. q ∈ Add(a_{add})

If no such action a_{add} exists, **Fail**

Q' = Q ∪ (a_{add},q,a_{need})

O' = O ∪ (a_{add} < a_{need})

agenda' = agenda - (q,a_{need})

If a_{add} is new, then A = A ∪ a_{add} and

∀ p ∈ PC(a_{add}), add (p, a_{add}) to agenda'

For every action a_t that threatens any causal link (a_i, Q, a_j) in Q'

Choose to add a_t < a_i or a_j < a_t to O

if neither choice is consistent, **Fail**

POP((A',O',Q'), agenda, actions)

POP terminates