CS 4649/7649 RIP
Robot Intelligence: Planning

Partial Order Planning, Graphplan

Sungmoon Joo

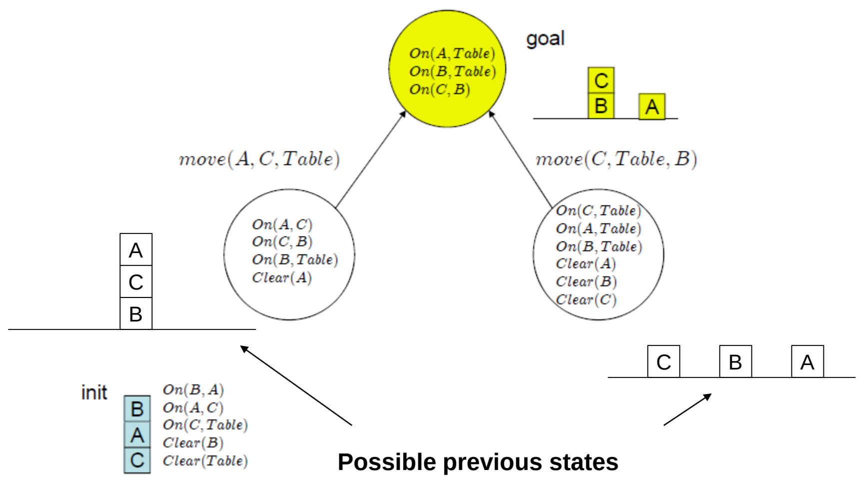
School of Interactive Computing
College of Computing
Georgia Institute of Technology

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

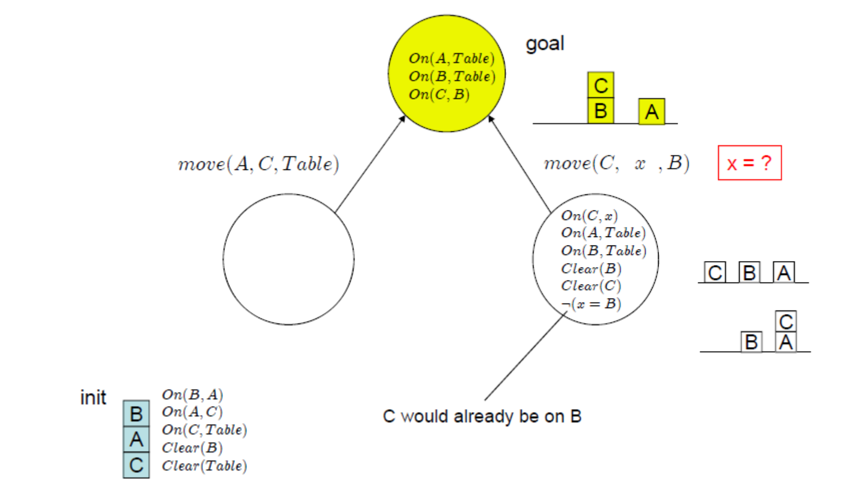
Course Info.

- **Course Website: joosm.github.io/RIP2014**
- **Course Wiki: github.com/RIP2014/RIP2014Wiki/wiki**
  - add your contact info, start grouping/filling in project ideas, etc.
  - github invitation sent (if you didn't get one, let me know)
  *announcements on t-square
- **Email me(sungmoon.joo@cc.gatech.edu)**
  - Introduce yourself, your github id
  - Project ideas, etc.

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"

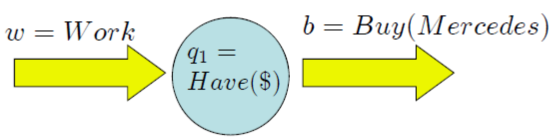
Make choices **only** when they are relevant to solving
The current part of the problem

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

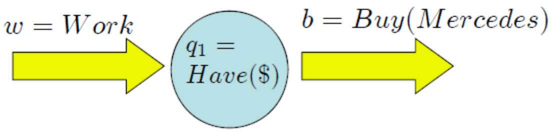
- A threat to (a,q,b) is defined as an action **t** that:
  - Has ¬ q as an effect

    $\neg q \in t_{add}$

  - 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 ∈ 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)

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 ∈ 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'

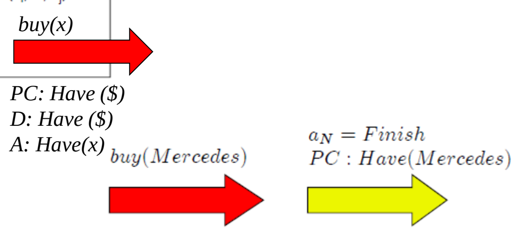
For every action a_t that threatens any causal link (a_i, Q, a_j) in Q'
Choose to add a_t < a_i or a_j < a_t to O
if neither choice is consistent, **Fail**

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

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'

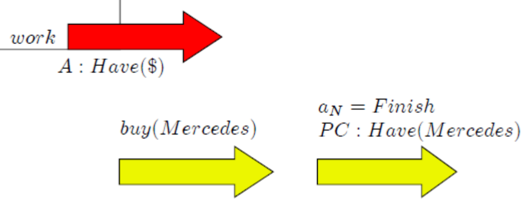
For every action a_t that threatens any causal link (a_i, Q, a_j) in Q'
Choose to add $a_t < a_i$ or $a_j < a_t$ to O
if neither choice is consistent, **Fail**

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'

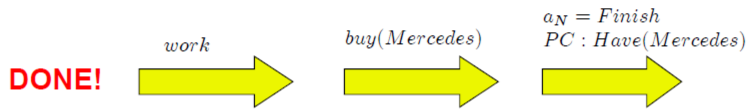
For every action a_t that threatens any causal link (a_i, Q, a_j) in Q'
Choose to add $a_t < a_i$ or $a_j < a_t$ to O
if neither choice is consistent, **Fail**

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

Agenda ={
}

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

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



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. $q \in Add(a_{add})$
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)

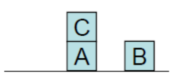
POP terminates

POP solves Sussman Anomaly

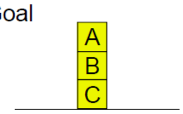
A_0

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

Init



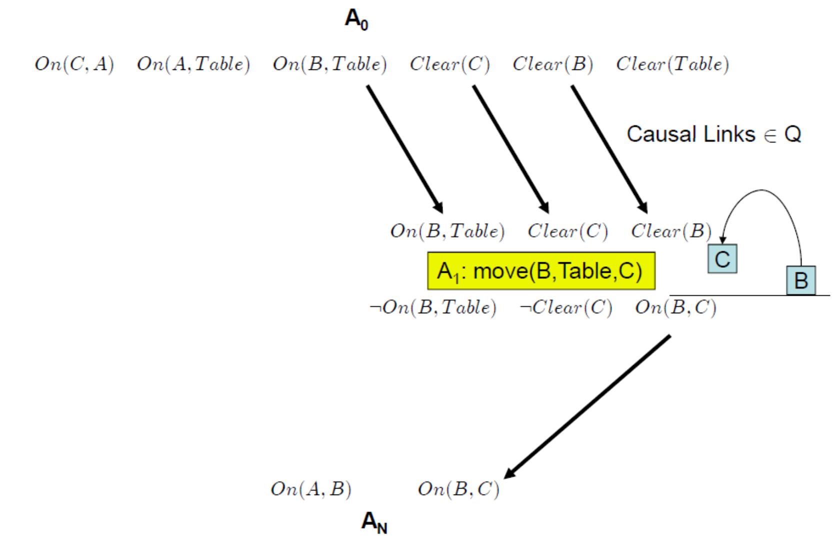
Goal



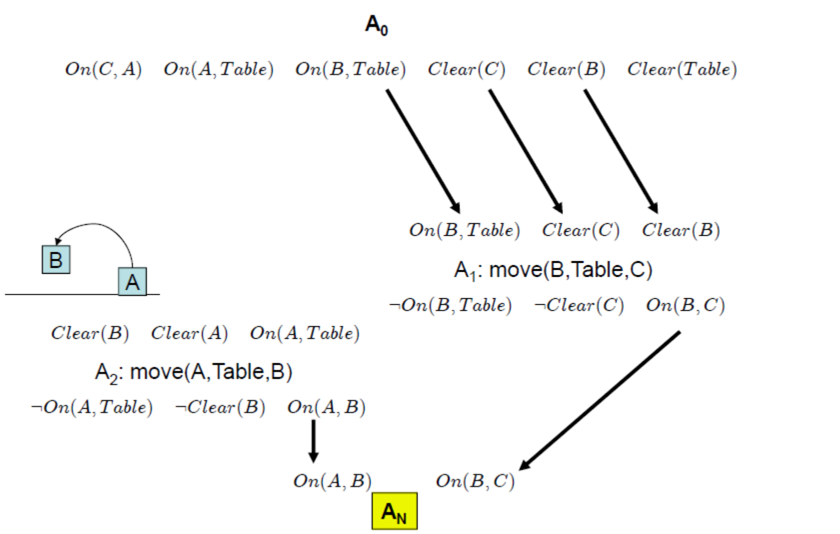
$On(A,B)$ $On(B,C)$

A_N

POP solves Sussman Anomaly



POP solves Sussman Anomaly



POP Algorithm Details

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'

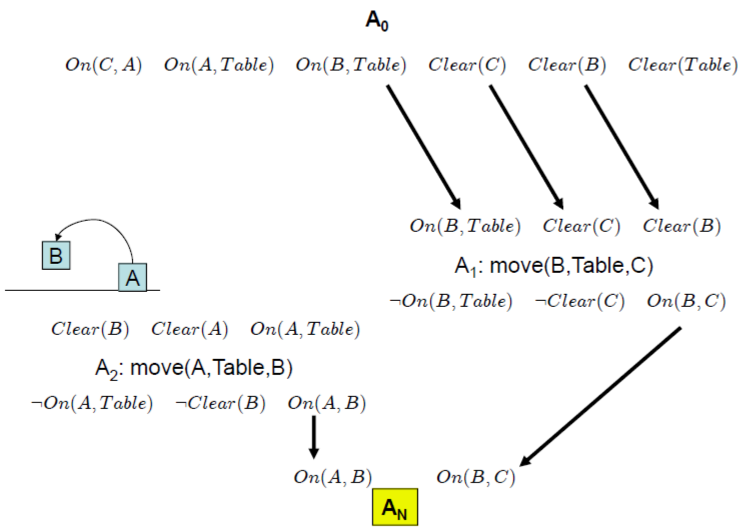
For every action a_t that **threatens** any causal link (a_i, Q, a_j) in Q'

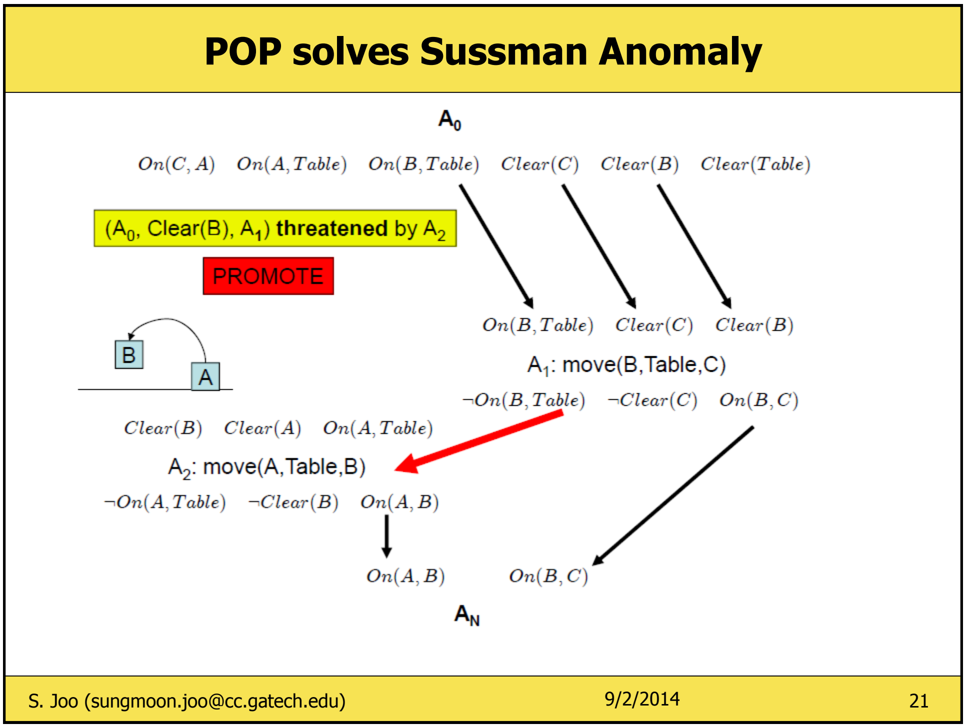
Choose to add $a_t < a_i$ or $a_j < a_t$ to O

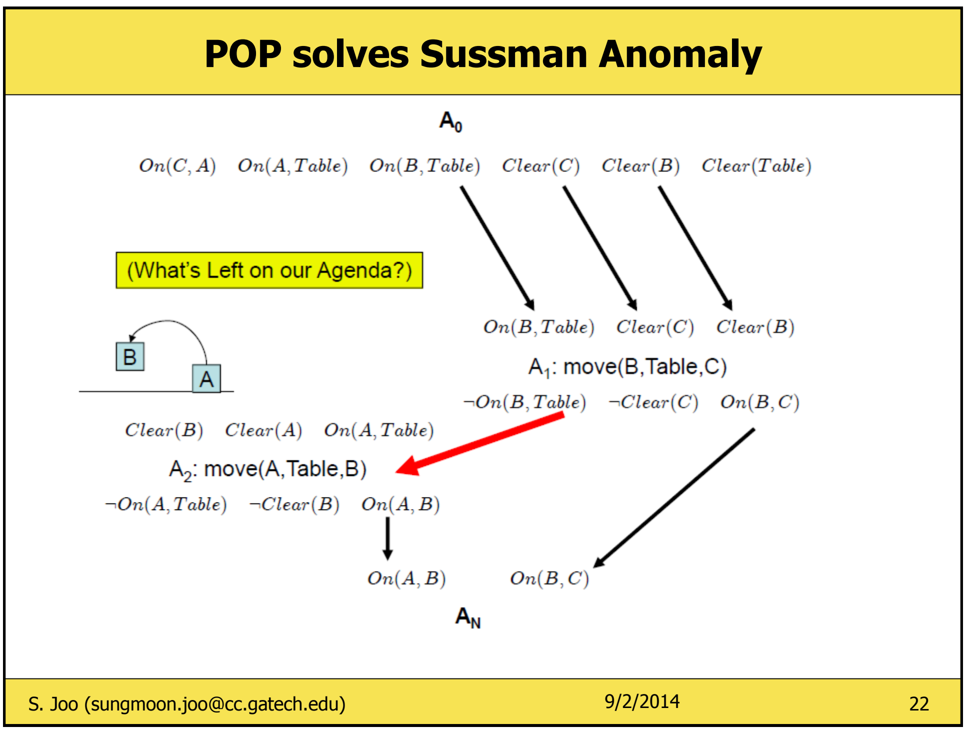
if neither choice is consistent, **Fail**

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

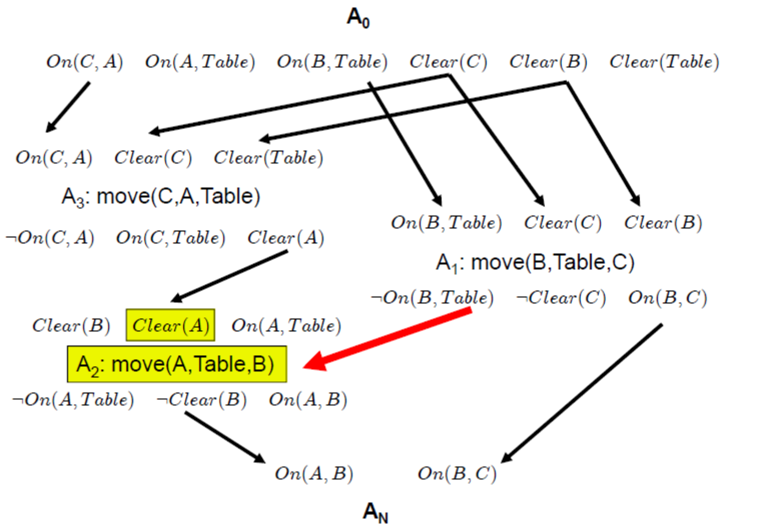
POP solves Sussman Anomaly



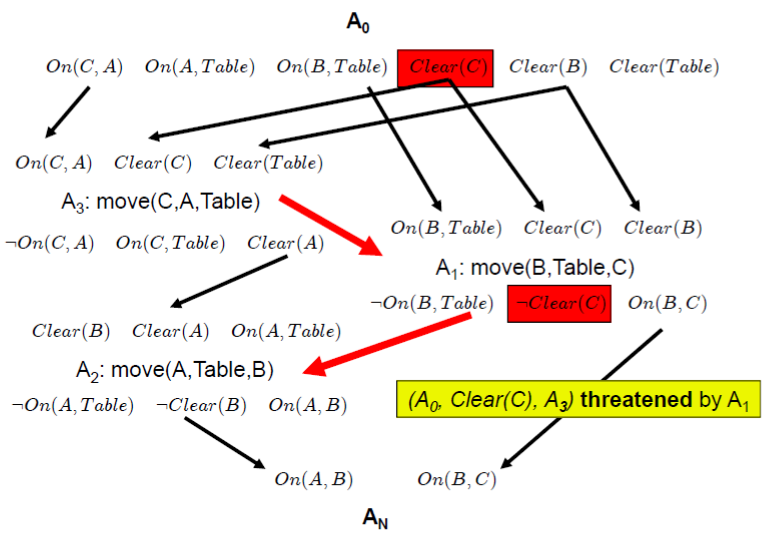

POP solves Sussman Anomaly
A0
On(C, A) On(A, Table) On(B, Table) Clear(C) Clear(B) Clear(Table)
(A0, Clear(B), A1) threatened by A2
PROMOTE
B
A
On(B, Table) Clear(C) Clear(B)
A1: move(B,Table,C)
¬On(B, Table) ¬Clear(C) On(B, C)
Clear(B) Clear(A) On(A, Table)
A2: move(A,Table,B)
¬On(A, Table) ¬Clear(B) On(A, B)
On(A, B) On(B, C)
AN
S. Joo (sungmoon.joo@cc.gatech.edu)
9/2/2014
21


POP solves Sussman Anomaly
A0
On(C, A) On(A, Table) On(B, Table) Clear(C) Clear(B) Clear(Table)
(What's Left on our Agenda?)
B
A
On(B, Table) Clear(C) Clear(B)
A1: move(B,Table,C)
¬On(B, Table) ¬Clear(C) On(B, C)
Clear(B) Clear(A) On(A, Table)
A2: move(A,Table,B)
¬On(A, Table) ¬Clear(B) On(A, B)
On(A, B) On(B, C)
AN
S. Joo (sungmoon.joo@cc.gatech.edu)
9/2/2014
22

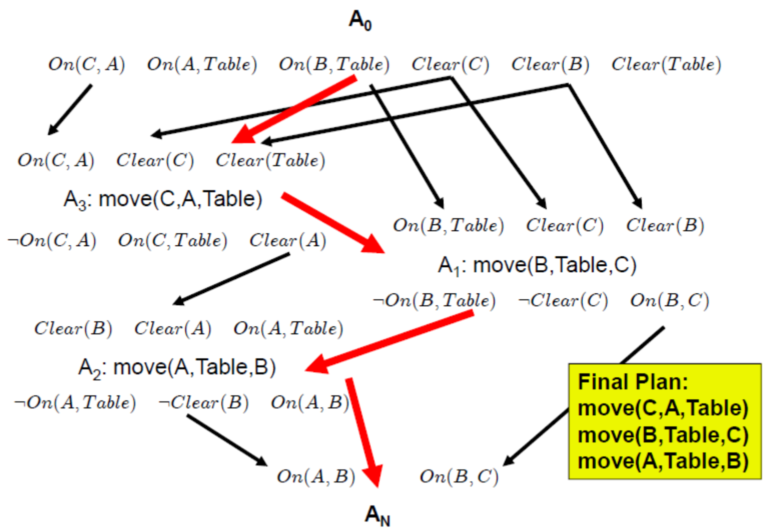
POP solves Sussman Anomaly



POP solves Sussman Anomaly



POP solves Sussman Anomaly



Properties of a Planner

1) Sound: The planner produces valid plans

2) Optimal: The planner produces optimal (shortest) plans

3) Complete: The planner finds a solution when there is one or returns that the solution is not possible.

POP Algorithm Details

Add Helpful Actions

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. $q \in Add(a_{add})$
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)

Recursion

Summary

- Important Properties of Planners
  - Soundness
  - Completeness
  - Optimality

Two types of planning:

- State Space
  - Non-deterministic Choices: n = |actions|
  - Backtracking for goal ordering
  - Simplicity!

- Plan Space
  - Least Commitment
  - Non-deterministic Choices: n = |preconditions| + |link protection|
  - Smaller branching factor (goal ordering not relevant)
  - Typically more optimal

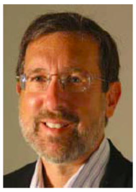
State Space vs. Plan Space

	State Space	Plan Space
Algorithm	Progression Regression	Partial Order Planning
Nodes	World States	Partial Plans
Transitions	Actions -Move(x,y,z) -Load(x,y) -Open(r)	Plan Refinement Ops -Adding Steps -Promotion -Demotion

Last Major Classical Planner: Graphplan*

- A form of state space planning: actually "Factored State Space"
- Nodes are **literals** and **instantiated actions**
- Uses Least Commitment principle to constrain search
- Simpler than Plan Space Planning
- More general than State Space Planning

Avrim Blum



Merrick Furst

1997

http://www.cs.cmu.edu/~avrim/graphplan.html

Graphplan Basics

- Graph has levels that represent time steps
- Levels alternate between preconditions and actions
  P_0 A_0 P_1 A_1 P_2 A_2 …
- Precondition level N reflects what **could** be true in N steps
- Mutex Links indicate that actions / preconditions **cannot** occur together.

Mutual Exclusion encodes the constraints on the action space.

Graph Nodes: P_i – Literals, A_i Actions

Add an action in level A_i if all of its preconditions are in P_i

Add a precondition in P_{i+1} if it is the effect of some action in A_i

Mutex Actions:
- One "clobbers" the other's effects or preconditions
- They have Mutex preconditions

Mutex Preconditions:
- All ways of achieving them are mutex

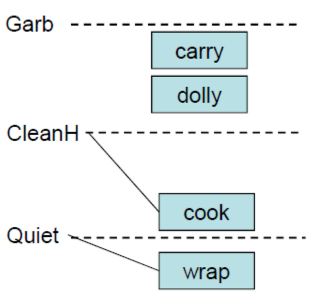
Mutual Exclusion encodes the constraints on the action space.

Example: Dinner Date

- Init: Garbage CleanHands Quiet
- Goal: Dinner Present ¬ Garbage
- Actions:

cook	PC: CleanHands	EF: Dinner
wrap	PC: Quiet	EF: Present
carry	PC:	EF: ¬ Garbage ¬ CleanHands
dolly	PC:	EF: ¬ Garbage ¬ Quiet

Initial Actions: A_0

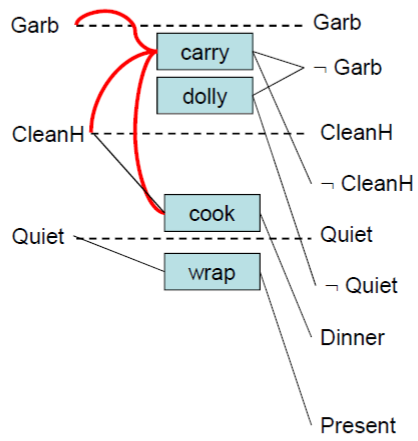


All actions whose preconditions are satisfied

Init: Garbage CleanHands Quiet
Goal: Dinner Present ¬ Garbage
Actions:

cook	PC: CleanHands	EF: Dinner
wrap	PC: Quiet	EF: Present
carry	PC:	EF: ¬ Garbage ¬ CleanHands
dolly	PC:	EF: ¬ Garbage ¬ Quiet

A_0 Mutex: CARRY Action



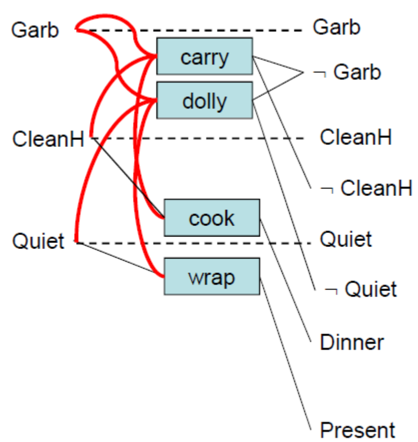
Mutex Actions:

- One "clobbers" the other's effects or preconditions

- They have Mutex preconditions

Init:	Garbage	CleanHands	Quiet
Goal:	Dinner	Present	¬ Garbage
Actions:			
cook	PC: CleanHands		EF: Dinner
wrap	PC: Quiet	EF: Present	
carry	PC:	EF: ¬ Garbage	¬ CleanHands
dolly	PC:	EF: ¬ Garbage	¬ Quiet

A_0 Mutex: All



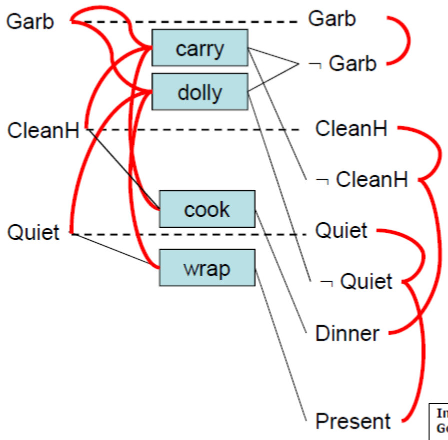
Mutex Actions:

- One "clobbers" the other's effects or preconditions

- They have Mutex preconditions

Init:	Garbage	CleanHands	Quiet
Goal:	Dinner	Present	¬ Garbage
Actions:			
cook	PC: CleanHands		EF: Dinner
wrap	PC: Quiet	EF: Present	
carry	PC:	EF: ¬ Garbage	¬ CleanHands
dolly	PC:	EF: ¬ Garbage	¬ Quiet

A_0 Mutex: Complete



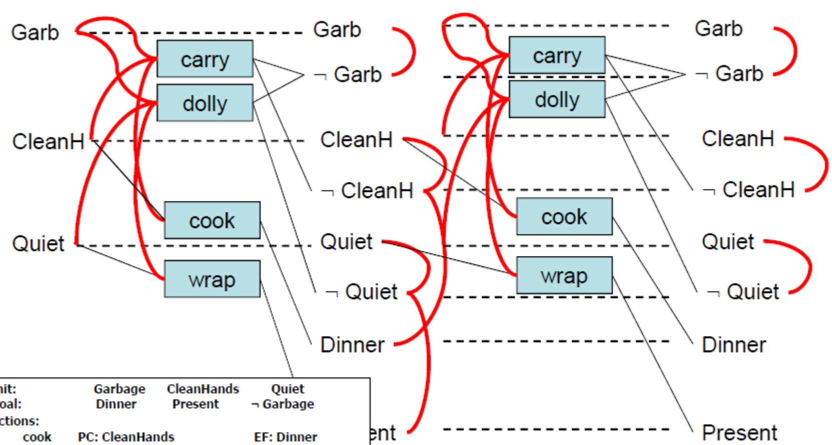
Mutex Actions:

- One "clobbers" the other's effects or preconditions

- They have Mutex preconditions

Init:	Garbage	CleanHands	Quiet
Goal:	Dinner	Present	¬ Garbage
Actions:			
cook	PC: CleanHands		EF: Dinner
wrap	PC: Quiet	EF: Present	
carry	PC:	EF: ¬ Garbage	¬ CleanHands
dolly	PC:	EF: ¬ Garbage	¬ Quiet

A_1: Action P_2: Precondition

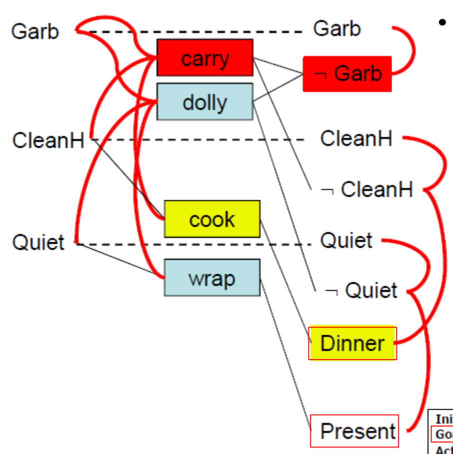


Init:	Garbage	CleanHands	Quiet
Goal:	Dinner	Present	¬ Garbage
Actions:			
cook	PC: CleanHands		EF: Dinner
wrap	PC: Quiet	EF: Present	
carry	PC:	EF: ¬ Garbage	¬ CleanHands
dolly	PC:	EF: ¬ Garbage	¬ Quiet

When can we make a plan?

- All goal conditions are satisfied
- Actions at the same level don't interfere
- Each action's preconditions are made true
- Is this sufficient?
  - No, but it's necessary
- Regression search from goals to check for Mutex Conditions

Solution at Level 1?



- **Build a graph**
- **Extract a solution(Backward Search)**

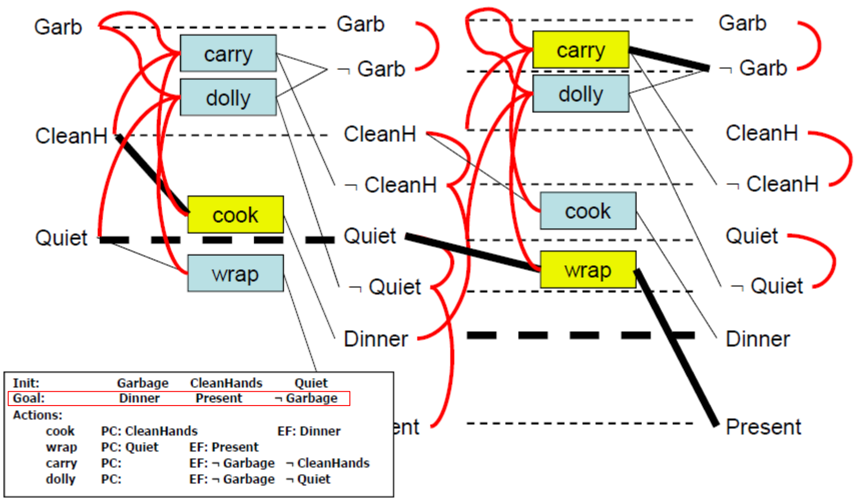
Repeat until goal set is empty
If goals are present & non-mutex:
1) Choose set of non-mutex actions to achieve each goal
2) Add preconditions to next goal set

Tow possible sets of actions at level 1:
{wrap, cook, dolly}
{wrap, cook, carry}

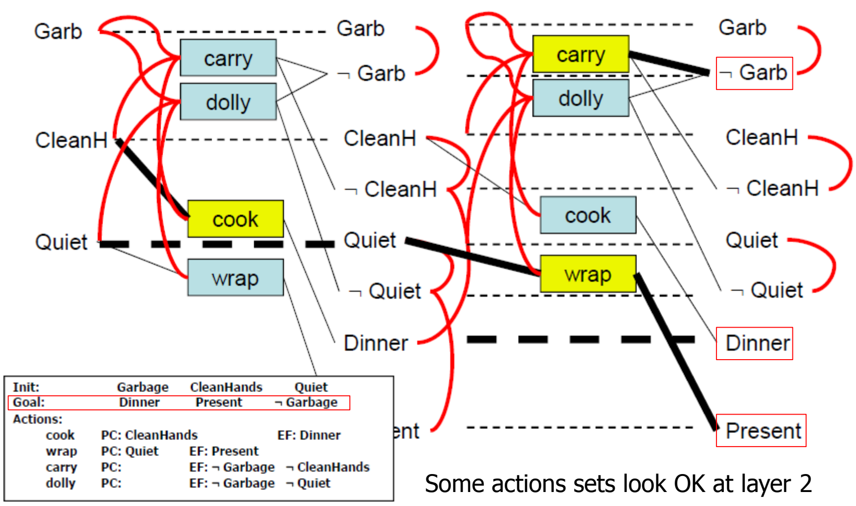
Neither set works – why?

Init:		**Garbage**	**CleanHands**	**Quiet**
Goal:		**Dinner**	**Present**	**¬ Garbage**
Actions:				
cook	**PC: CleanHands**			**EF: Dinner**
wrap	**PC: Quiet**	**EF: Present**		
carry	**PC:**	**EF: ¬ Garbage**	**¬ CleanHands**	
dolly	**PC:**	**EF: ¬ Garbage**	**¬ Quiet**	

Add new layer

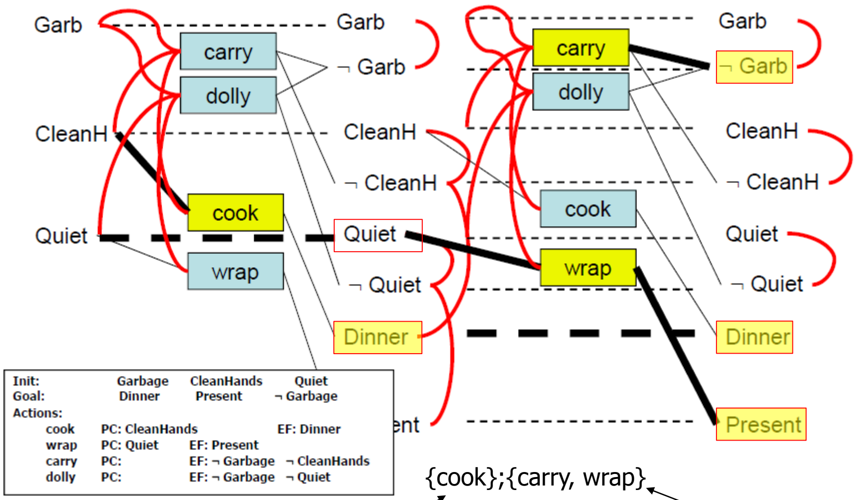


Solution at Level 2?



Some actions sets look OK at layer 2

Solution at Level 2?

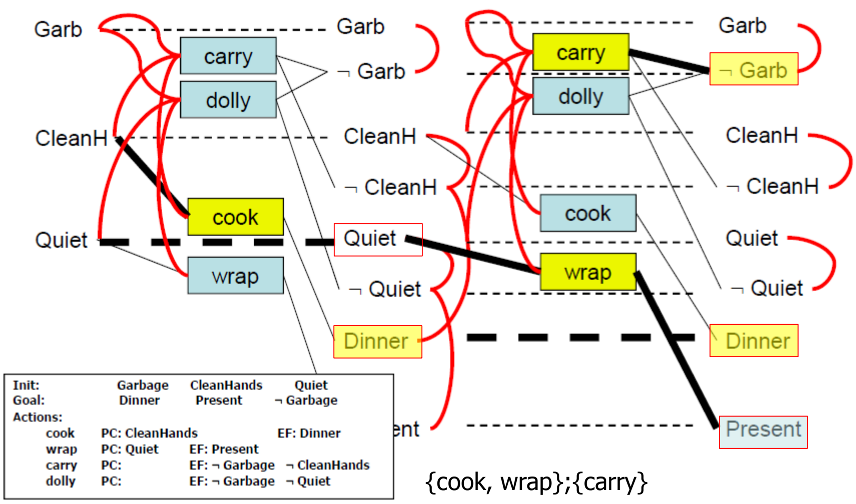


{cook};{carry, wrap}

Possible action at level 2

Possible action at level 1

Solution at Level 2?



{cook, wrap};{carry}

Another possible action sets

Observations

- Actions monotonically increase
- Propositions monotonically increase
- Mutex Conditions monotonically decrease

Graphplan Conclusions

- Combines simplicity of state-space planning with constraints of partial order planning
- Very popular method for extending and evaluating improvements

State Space vs. Plan Space vs. Graph Space

	State Space (60s –)	Plan Space (70s –)	Graph Space (90s -)
Algorithm	Progression Regression	Partial Order Planning	Iterates Graph Building Regression Search
Nodes	World States	Partial Plans	Graph Levels
Transitions	Actions -Move(x,y,z) -Load(x,y) -Open(r)	Plan Refinement Operations -Adding Steps -Promotion -Demotion	Sets of Actions -Constraint Coding -Mutual Exclusion

Modern Action Representation

Move ?b from ?x to ?y
parameters: ?b, ?x, ?y
preconditions: (and (on ?b ?x) (clear ?b) (clear ?y)
(≠ ?b ?x) (≠ ?b ?y) (≠ ?x ?y)
(≠ ?y Table))
effects: (and (on ?b ?y) (not (on ?b ?x)) (clear ?x) (not (clear ?y)))

PDDL: Planning Domain Definition Language*

Expressive Actions

- Actions with variables
- Disjunctive preconditions
- Conditional effects
- Universal quantification

UCPOP [Penberthy and Weld'92]

Action Representation

```
Move ?b from ?x to ?y
    parameters: ?b, ?x, ?y
    preconditions: (and (on ?b ?x) (clear ?b) (clear ?y)
                        (≠ ?b ?x) (≠ ?b ?y) (≠ ?x ?y)
                        (≠ ?y Table) )
    effects:  (and (on ?b ?y) (not (on ?b ?x)) (clear ?x) (not (clear ?y)))
```

VS

Conditional Effect

```
Move ?b from ?x to ?y
    parameters: ?b, ?x, ?y
    preconditions: (and (on ?b ?x) (clear ?b) (clear ?y)
                        (≠ ?b ?x) (≠ ?b ?y) (≠ ?x ?y)
    effects:  (and (on ?b ?y) (not (on ?b ?x)) (clear ?x)
                                  (when (≠ ?y Table) (not (clear ?y))) )
```

PDDL: Planning Domain Definition Language*

Conditional Planning

- Conditional Effects
  - Effects that depend on state
  - Require special attention but same planning concept
- **Disjunctive Effects**
  - Different concept entirely
  - We don't know the outcome of an action