

CS 561: Artificial Intelligence

Instructor: Sofus A. Macskassy, macskass@usc.edu

TAs: Nadeesha Ranashinghe (nadeeshr@usc.edu)

William Yeoh (wyeoh@usc.edu)

Harris Chiu (chiciu@usc.edu)

Lectures: MW 5:00-6:20pm, OHE 122 / DEN

Office hours: By appointment

Class page: <http://www-rcf.usc.edu/~macskass/CS561-Spring2010/>

This class will use <http://www.uscden.net/> and class webpage

- Up to date information
- Lecture notes
- Relevant dates, links, etc.

Course material:

[AIMA] Artificial Intelligence: A Modern Approach,
by Stuart Russell and Peter Norvig. (2nd ed)

Planning (AIMA Ch. 11)

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

What we have so far

- Can TELL KB about new percepts about the world
- KB maintains model of the current world state
- Can ASK KB about any fact that can be inferred from KB

How can we use these components to build a **planning agent**?

i.e., an agent that constructs plans that can achieve its goals,
and that then executes these plans?

Remember: Problem-Solving Agent

```
function SIMPLE-PROBLEM-SOLVING-AGENT( percept) returns an action
static:
    seq, an action sequence, initially empty
    state, some description of the current world state
    goal, a goal, initially null
    problem, a problem formulation

state ← UPDATE-STATE(state, percept) // What is the current state?
if seq is empty then
    goal ← FORMULATE-GOAL(state) // From LA to San Diego (given curr. state)
    problem ← FORMULATE-PROBLEM(state, goal) // e.g., Gas usage
    seq ← SEARCH(problem)
action ← RECOMMENDATION(seq, state)
seq ← REMAINDER(seq, state) // If fails to reach goal, update
return action
```

Note: This is *offline* problem-solving. *Online* problem-solving involves acting w/o complete knowledge of the problem and environment

Simple planning agent

- Use percepts to build model of current world state
- IDEAL-PLANNER: Given a goal, algorithm generates plan of action
- STATE-DESCRIPTION: given percept, return initial state description in format required by planner
- MAKE-GOAL-QUERY: used to ask KB what next goal should be

A Simple Planning Agent

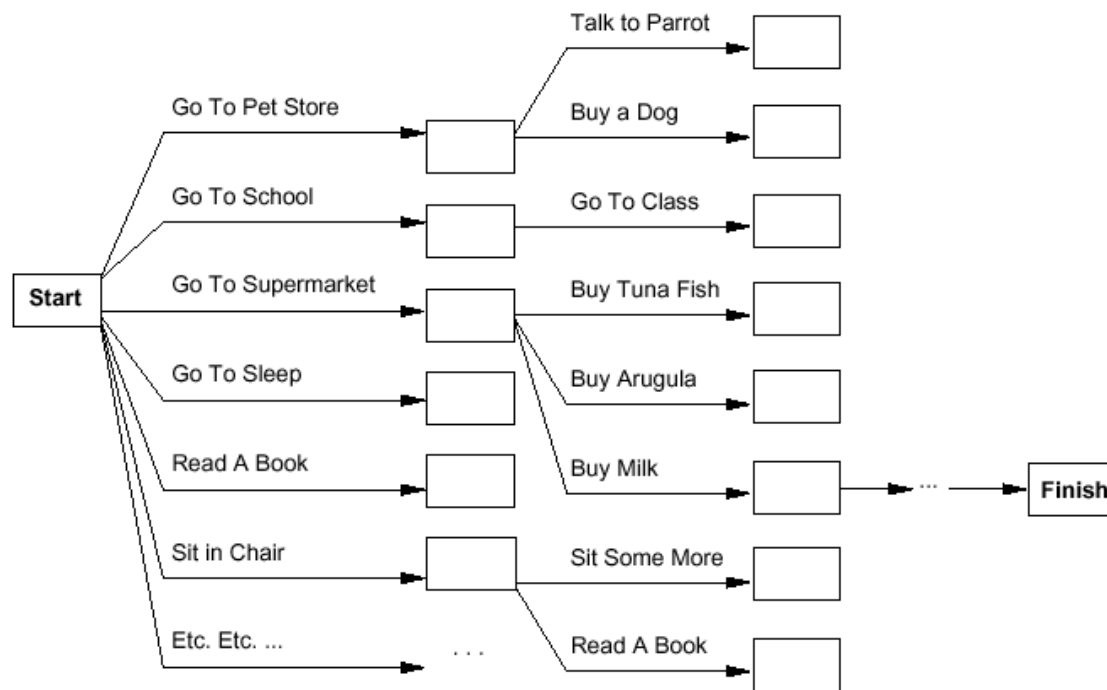
```
function SIMPLE-PLANNING-AGENT(percept) returns an action
static:
    KB, a knowledge base (includes action descriptions)
    p, a plan (initially, NoPlan)
    t, a time counter (initially 0)
local variables:G, a goal
    current, a current state description
TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
current ← STATE-DESCRIPTION(KB, t)
if p = NoPlan then
    G ← ASK(KB, MAKE-GOAL-QUERY(t))
    p ← IDEAL-PLANNER(current, G, KB)
if p = NoPlan or p is empty then
    action ← NoOp
else
    action ← FIRST(p)
    p ← REST(p)
TELL(KB, MAKE-ACTION-SENTENCE(action, t))
t ← t+1
return action
```

Like popping from a stack

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

Search vs. planning

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

	Search	Planning
States	Lisp data structures	Logical sentences
Actions	Lisp code	Preconditions/outcomes
Goal	Lisp code	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints on actions

Planning in situation calculus

$PlanResult(p, s)$ is the situation resulting from executing p in s

$$PlanResult([], s) = s$$

$$PlanResult([a|p], s) = PlanResult(p, Result(a, s))$$

Initial state $At(Home, S_0) \wedge \neg Have(Milk, S_0) \wedge \dots$

Actions as Successor State axioms

$$Have(Milk, Result(a, s)) \Leftrightarrow$$

$$[(a = Buy(Milk) \wedge At(Supermarket, s)) \vee (Have(Milk, s) \wedge a \neq \dots)]$$

Query

$$s = PlanResult(p, S_0) \wedge At(Home, s) \wedge Have(Milk, s) \wedge \dots$$

Solution

$$p = [Go(Supermarket), Buy(Milk), Buy(Bananas), Go(HWS), \dots]$$

Principal difficulty: unconstrained branching, hard to apply heuristics

Basic representation for planning

- Most widely used approach: uses STRIPS language
- **states**: conjunctions of function-free ground literals (I.e., predicates applied to constant symbols, possibly negated); e.g.,

$\text{At}(\text{Home}) \wedge \neg\text{Have}(\text{Milk}) \wedge \neg\text{Have}(\text{Bananas}) \wedge \neg\text{Have}(\text{Drill}) \dots$

- **goals**: also conjunctions of literals; e.g.,

$\text{At}(\text{Home}) \wedge \text{Have}(\text{Milk}) \wedge \text{Have}(\text{Bananas}) \wedge \text{Have}(\text{Drill})$

but can also contain variables (implicitly universally quant.); e.g.,

$\text{At}(x) \wedge \text{Sells}(x, \text{Milk})$

Planner vs. theorem prover

- **Planner:** ask for sequence of actions that makes goal true if executed
- **Theorem prover:** ask whether query sentence is true given KB

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!]

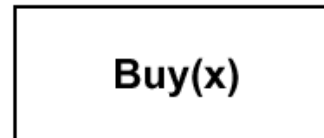
Restricted language \Rightarrow efficient algorithm

Precondition: conjunction of positive literals

Effect: conjunction of literals

Graphical notation:

$At(p) Sells(p, x)$



$Have(x)$

STRIPS cont'd

- Every literal not mentioned in effect remains unchanged
 - Circumvents the frame problem mentioned earlier

STRIPS vs. Action Desc. Lang (ADL)

STRIPS is not always adequate for all real-world domains and other planning languages have been developed to address some deficiencies

STRIPS Language	ADL Language
Only positive literals in state: Poor \wedge Unknown	Positive and negative literals \neg Rich \wedge \neg Famous
Closed World Assumption: Unmentioned literals are false	Open World Assumption: Unmentioned literals are unknown
Effect $P \wedge \neg Q$ means add P and delete Q	Effect $P \wedge \neg Q$ means add P and $\neg Q$ and delete $\neg P$ and Q
Only ground literals in goals	Quantified variables in goals
Goals are conjunctions	Goals allow disjuncts + conjunctions
Effects are conjunctions	Conditional effects allowed
No support for equality	Equality built in
No support for types	Variables can have types

STRIPS vs ADL cont'd

Comparison of 'fly' action definitions

STRIPS:

- Action(Fly(p,from,to),
Precond: $At(p,from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
Effect: $\sim At(p,from) \wedge At(p,to)$)

ADL:

- Action(Fly(p: Plane, from: Airport, to: Airport),
Precond: $At(p,from) \wedge (from \neq to)$
Effect: $\sim At(p,from) \wedge At(p,to)$)

Example STRIPS problem

- $\text{Init}(\text{At}(C1, \text{SFO}) \wedge \text{At}(C2, \text{JFK}) \wedge \text{At}(P1, \text{SFO}) \wedge \text{At}(P2, \text{JFK}) \wedge \text{Cargo}(C1) \wedge \text{Cargo}(C2) \wedge \text{Plane}(P1) \wedge \text{Plane}(P2) \wedge \text{Airport}(\text{JFK}) \wedge \text{Airport}(\text{SFO}))$
- $\text{Goal}(\text{At}(C1, \text{JFK}) \wedge \text{At}(C2, \text{SFO}))$
- $\text{Action}(\text{Load}(c, p, a),$
 - Precond: $\text{At}(c, a) \wedge \text{At}(p, a) \wedge \text{Cargo}(c) \wedge \text{Plane}(p) \wedge \text{Airport}(a)$
 - Effect: $\sim \text{At}(c, a) \wedge \text{In}(c, p)$
- $\text{Action}(\text{Unload}(c, p, a),$
 - Precond: $\text{In}(c, p) \wedge \text{At}(p, a) \wedge \text{Cargo}(c) \wedge \text{Plane}(p) \wedge \text{Airport}(a)$
 - Effect: $\text{At}(c, a) \wedge \sim \text{In}(c, p)$
- $\text{Action}(\text{Fly}(p, \text{from}, \text{to}),$
 - Precond: $\text{At}(p, \text{from}) \wedge \text{Plane}(p) \wedge \text{Airport}(\text{from}) \wedge \text{Airport}(\text{to})$
 - Effect: $\sim \text{At}(p, \text{from}) \wedge \text{At}(p, \text{to})$

Example STRIPS problem

- $\text{Init}(\text{At}(C1, \text{SFO}) \wedge \text{At}(C2, \text{JFK}) \wedge \text{At}(P1, \text{SFO}) \wedge \text{At}(P2, \text{JFK}) \wedge \text{Cargo}(C1) \wedge \text{Cargo}(C2) \wedge \text{Plane}(P1) \wedge \text{Plane}(P2) \wedge \text{Airport}(\text{JFK}) \wedge \text{Airport}(\text{SFO}))$
- $\text{Goal}(\text{At}(C1, \text{JFK}) \wedge \text{At}(C2, \text{SFO}))$

One possible solution:

- Action(Load(c,p)
 - Precond: $\text{At}(c, \text{SFO})$
 - Effect: $\sim \text{At}(c, \text{SFO}) \wedge \text{At}(c, p)$
- Action(Unload(c,p)
 - Precond: $\text{In}(c, p)$
 - Effect: $\text{At}(c, \text{SFO})$
- Action(Fly(p,from,to)
 - Precond: $\text{At}(p, \text{from})$
 - Effect: $\sim \text{At}(p, \text{from}) \wedge \text{At}(p, \text{to})$

Load(C1,P1,SFO)
Fly(P1,SFO,JFK)
Unload(C1,P1,JFK)

Load(C2,P2,JFK)
Fly(P2,JFK,SFO)
Unload(C2,P2,SFO)

a)
a)
to)

Types of planners

- Situation space planner: search through possible situations
- Progression planner: start with initial state, apply operators until goal is reached
 - Problem: high branching factor!
- Regression planner: start from goal state and apply operators until start state reached
 - Why desirable? usually many more operators are applicable to initial state than to goal state.
 - Difficulty: when want to achieve a conjunction of goals

Initial STRIPS algorithm: situation-space regression planner

State space vs. plan space

Standard search: node = concrete world state

Planning search: node = partial plan

Search space of plans rather than of states.

Defn: open condition is a precondition of a step not yet fulfilled

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

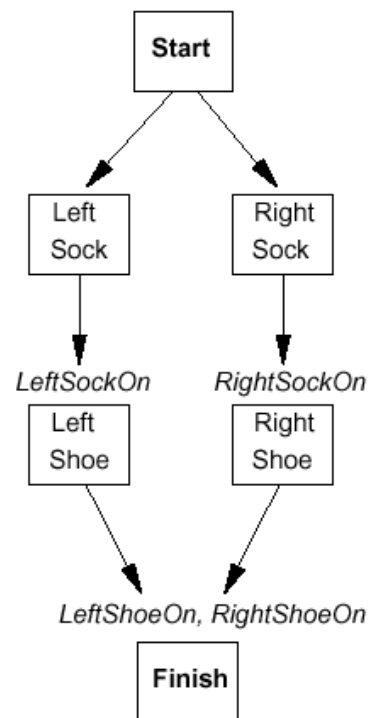
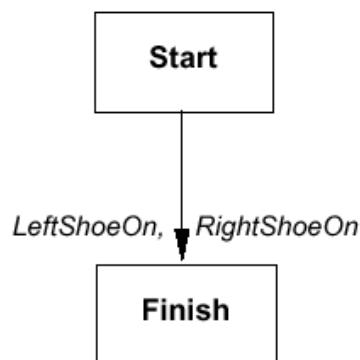
gradually move from incomplete/vague plans to complete, correct plans

Types of planners

- **Partial order planner:** some steps are ordered, some are not
- **Total order planner:** all steps ordered (thus, plan is a simple list of steps)
- **Linearization:** process of deriving a totally ordered plan from a partially ordered plan.

Partially ordered plans

How many total-order plans are there to get both shoes on?



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 STRIPS problem

- $\text{Init}(\text{At}(\text{C1}, \text{SFO}) \wedge \text{At}(\text{C2}, \text{JFK}) \wedge \text{At}(\text{P1}, \text{SFO}) \wedge \text{At}(\text{P2}, \text{JFK}) \wedge \text{Cargo}(\text{C1}) \wedge \text{Cargo}(\text{C2}) \wedge \text{Plane}(\text{P1}) \wedge \text{Plane}(\text{P2}) \wedge \text{Airport}(\text{JFK}) \wedge \text{Airport}(\text{SFO}))$
- $\text{Goal}(\text{At}(\text{C1}, \text{JFK}) \wedge \text{At}(\text{C2}, \text{SFO}))$

One possible solution:

```
[  
  Load(C1,P1,SFO)  
  Fly(P1,SFO,JFK)  
  Unload(C1,P1,JFK)  
  Load(C2,P2,JFK)  
  Fly(P2,JFK,SFO)  
  Unload(C2,P2,JFK)  
]
```

$(p,a) \wedge \text{Cargo}(c) \wedge \text{Plane}(p) \wedge \text{Airport}(a)$
 $\wedge \text{In}(c,p)$

$(p,a) \wedge \text{Cargo}(c) \wedge \text{Plane}(p) \wedge \text{Airport}(a)$
 $\wedge \text{In}(c,p)$

$\wedge \text{Plane}(p) \wedge \text{Airport}(\text{from}) \wedge \text{Airport}(\text{to})$
 $\wedge \text{At}(p,\text{to})$

Example STRIPS problem

- $\text{Init}(\text{At}(\text{C1}, \text{SFO}) \wedge \text{At}(\text{C2}, \text{JFK}) \wedge \text{At}(\text{P1}, \text{SFO}) \wedge \text{At}(\text{P2}, \text{JFK}) \wedge \text{Cargo}(\text{C1}) \wedge \text{Cargo}(\text{C2}) \wedge \text{Plane}(\text{P1}) \wedge \text{Plane}(\text{P2}) \wedge \text{Airport}(\text{JFK}) \wedge \text{Airport}(\text{SFO}))$
- $\text{Goal}(\text{At}(\text{C1}, \text{JFK}) \wedge \text{At}(\text{C2}, \text{SFO}))$

One possible solution:

[
Load(C1,P1,SFO)
Fly(P1,SFO,JFK)
Unload(C1,P1,JFK)
Load(C2,P2,JFK)
Fly(P2,JFK,SFO)
Unload(C2,P2,JFK)
]



One possible solution:

[
Load(C1,P1,SFO)
Fly(P1,SFO,JFK)
Unload(C1,P1,JFK)
Load(C2,P2,JFK)
Fly(P2,JFK,SFO)
Unload(C2,P2,JFK)
]

Plan

We formally define a plan as a **data structure consisting of:**

- Set of **plan steps** (each is an operator for the problem)
- Set of **step ordering constraints**

e.g., $A \preceq B$ means "A before B"

- Set of **variable binding constraints**

e.g., $v = x$ where v variable and x constant or other variable

- Set of **causal links**

e.g., $A \xrightarrow{c} B$ means "A achieves c for B"

POP algorithm sketch

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 (cont.)

procedure CHOOSE-OPERATOR($plan, operators, S_{need}, c$)

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 $S_{add} \xrightarrow{c} S_{need}$ to $LINKS(plan)$

add the ordering constraint $S_{add} \prec S_{need}$ 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_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in $LINKS(plan)$ **do**

choose either

Demotion: Add $S_{threat} \prec S_i$ to $ORDERINGS(plan)$

Promotion: Add $S_j \prec S_{threat}$ to $ORDERINGS(plan)$

if not $CONSISTENT(plan)$ **then fail**

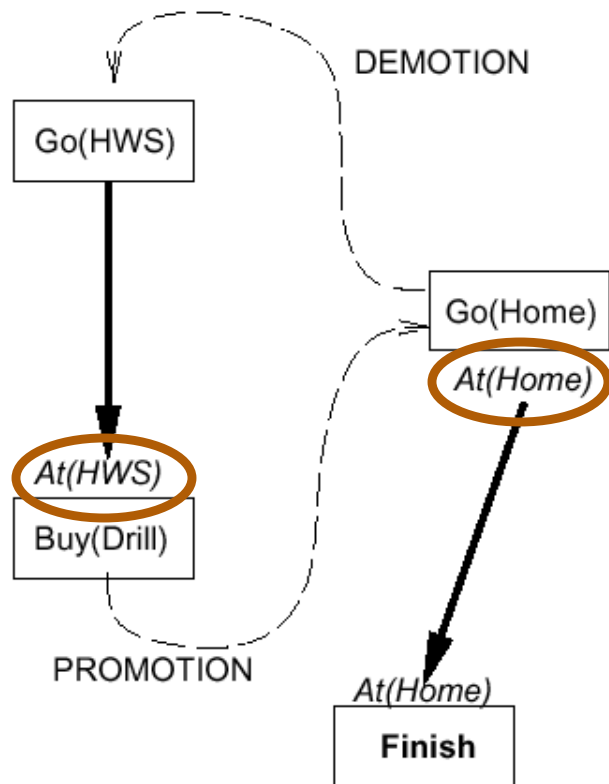
end

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

Extensions for disjunction, universals, negation, conditionals

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 $At(HWS)$:

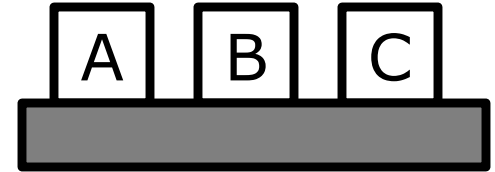


Demotion: put before $Go(HWS)$

Promotion: put after $Buy(Drill)$

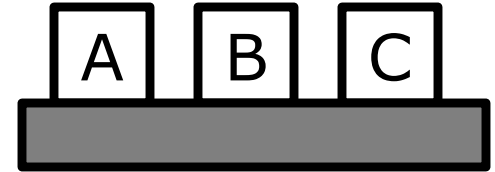
STRIPS: block world toy problem

- $\text{Init}(\text{On}(A, \text{Table}) \wedge \text{On}(B, \text{Table}) \wedge \text{On}(C, \text{Table})$
 $\wedge \text{Block}(A) \wedge \text{Block}(B) \wedge \text{Block}(C)$
 $\wedge \text{Clear}(A) \wedge \text{Clear}(B) \wedge \text{Clear}(C))$
- $\text{Goal}(\text{On}(A, B) \wedge \text{On}(B, C))$
- $\text{Action}(\text{PutOn}(x, y),$
 Precond: $\text{On}(x, z) \wedge \text{Clear}(x) \wedge \text{Clear}(y) \wedge \text{Block}(x) \wedge \text{Block}(y) \wedge$
 $(x \neq z) \wedge (x \neq y) \wedge (y \neq z)$
 Effect: $\text{On}(x, y) \wedge \text{Clear}(z) \wedge \sim \text{On}(x, z) \wedge \sim \text{Clear}(y))$
- $\text{Action}(\text{PutOnTable}(x),$
 Precond: $\text{On}(x, z) \wedge \text{Clear}(x) \wedge \text{Block}(x) \wedge (x \neq z)$
 Effect: $\text{On}(x, \text{Table}) \wedge \text{Clear}(z) \wedge \sim \text{On}(x, z)$



STRIPS: block world toy problem

- $\text{Init}(\text{On}(A, \text{Table}) \wedge \text{On}(B, \text{Table}) \wedge \text{On}(C, \text{Table})$
 $\wedge \text{Block}(A) \wedge \text{Block}(B) \wedge \text{Block}(C)$
 $\wedge \text{Clear}(A) \wedge \text{Clear}(B) \wedge \text{Clear}(C))$



- $\text{Goal}(\text{On}(A, B) \wedge \text{On}(B, C))$

- $\text{Action}(\text{PutOn}(x, y))$
 $\text{Precond: } \text{On}(x, z) \wedge \text{Clear}(x) \wedge \text{Block}(y) \wedge (x \neq y) \wedge (y \neq z)$
 $\text{Effect: } \text{On}(x, y) \wedge \text{Clear}(z) \wedge \sim \text{On}(x, z)$

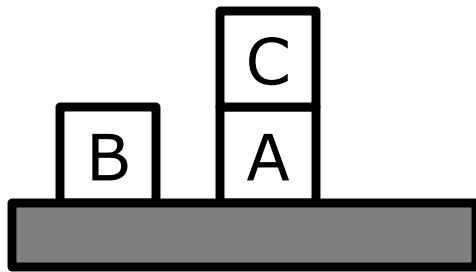
One possible solution:

PutOn(B,C)
PutOn(A,B)

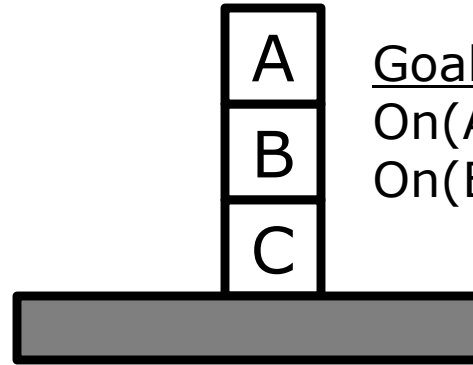
- $\text{Action}(\text{PutOnTable}(x))$
 $\text{Precond: } \text{On}(x, z) \wedge \text{Clear}(x) \wedge \text{Block}(x) \wedge (x \neq z)$
 $\text{Effect: } \text{On}(x, \text{Table}) \wedge \text{Clear}(z) \wedge \sim \text{On}(x, z)$

Order is very important!
Cannot do "PutOn(A,B)" first...!

"Sussman anomaly" problem



Start State



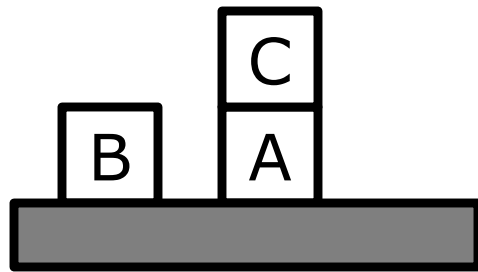
Goal State

Goal Conditions:

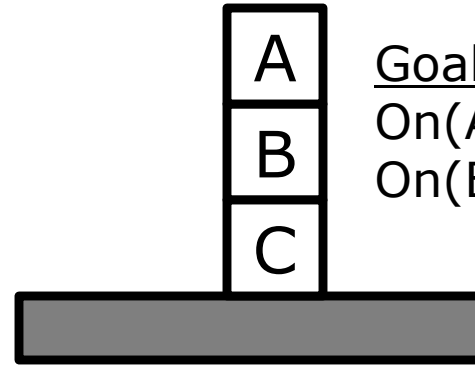
$\text{On}(A,B)$

$\text{On}(B,C)$

"Sussman anomaly" problem (cont.)



Start State



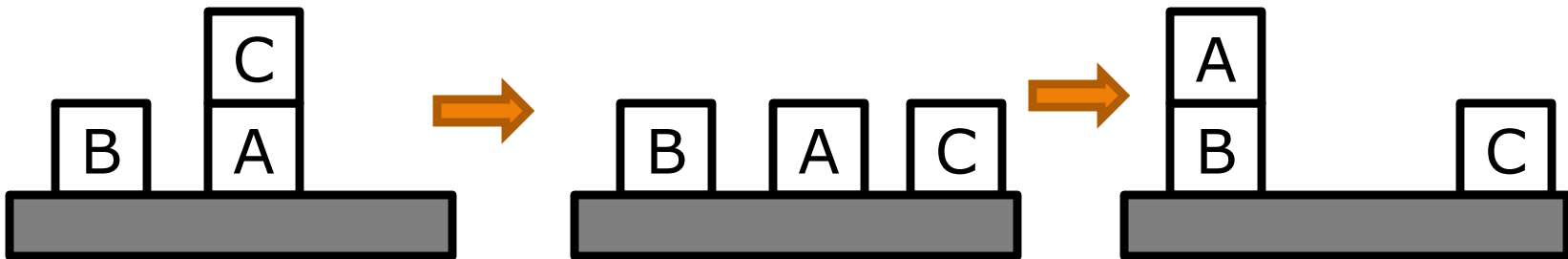
Goal State

Goal Conditions:

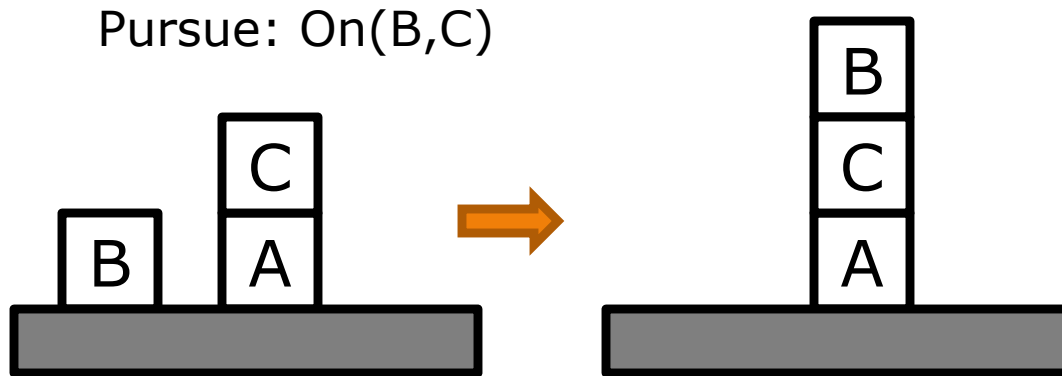
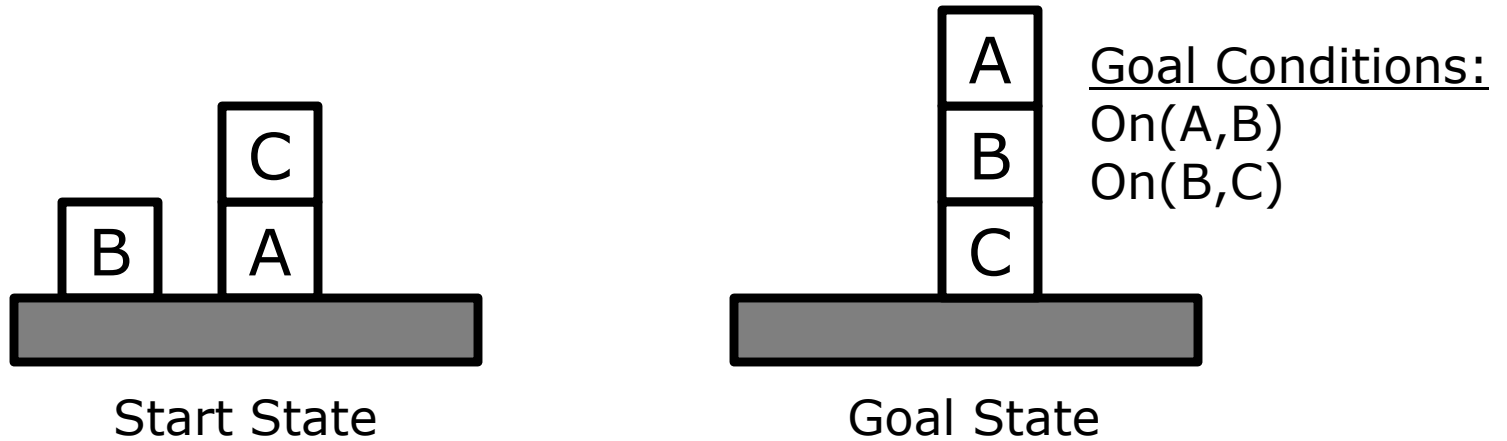
$\text{On}(A,B)$

$\text{On}(B,C)$

Pursue: $\text{On}(A,B)$



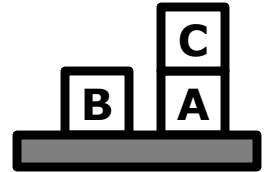
"Sussman anomaly" problem (cont.)



POP Solution... (cont.)

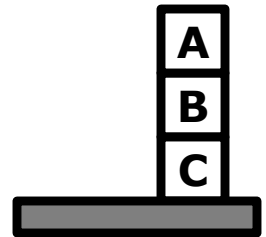
START

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

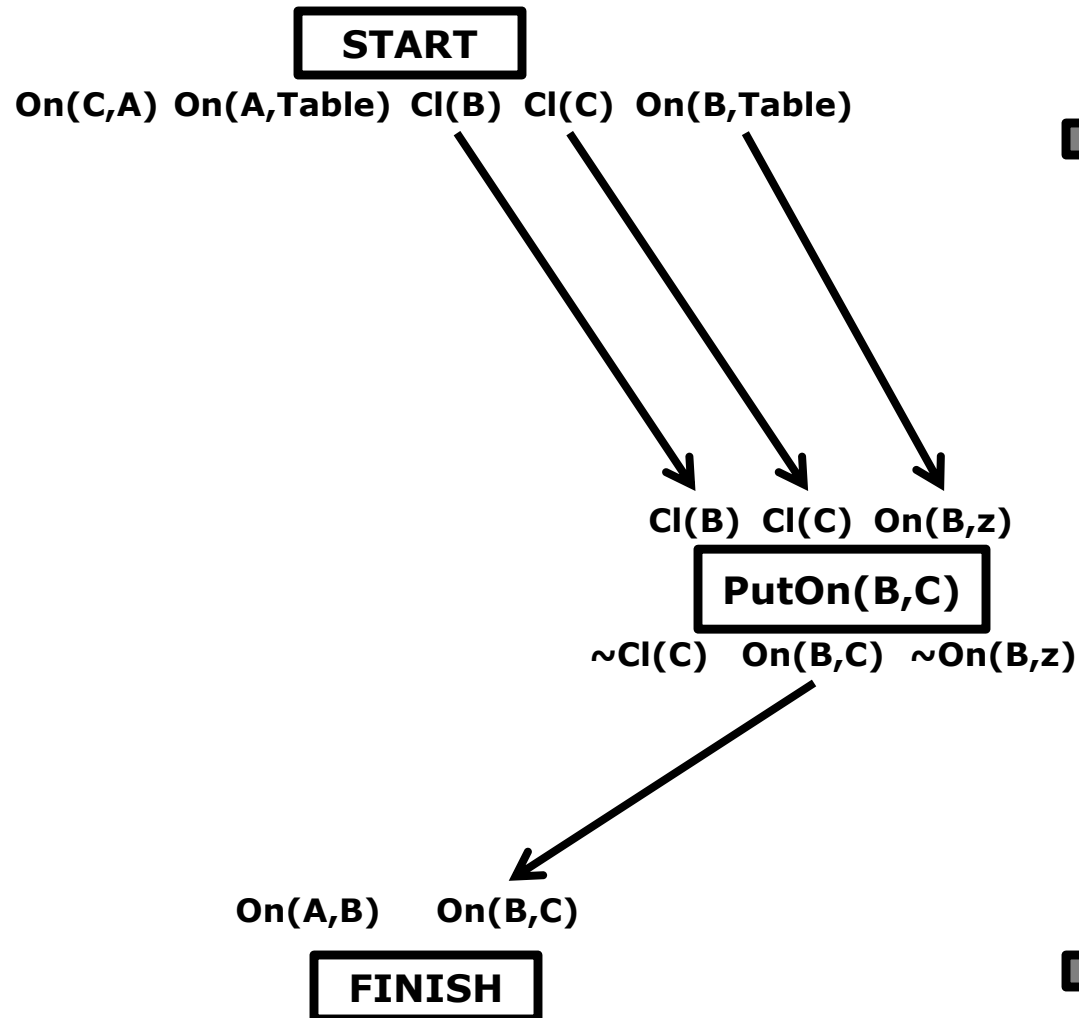


On(A,B) On(B,C)

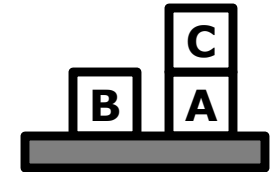
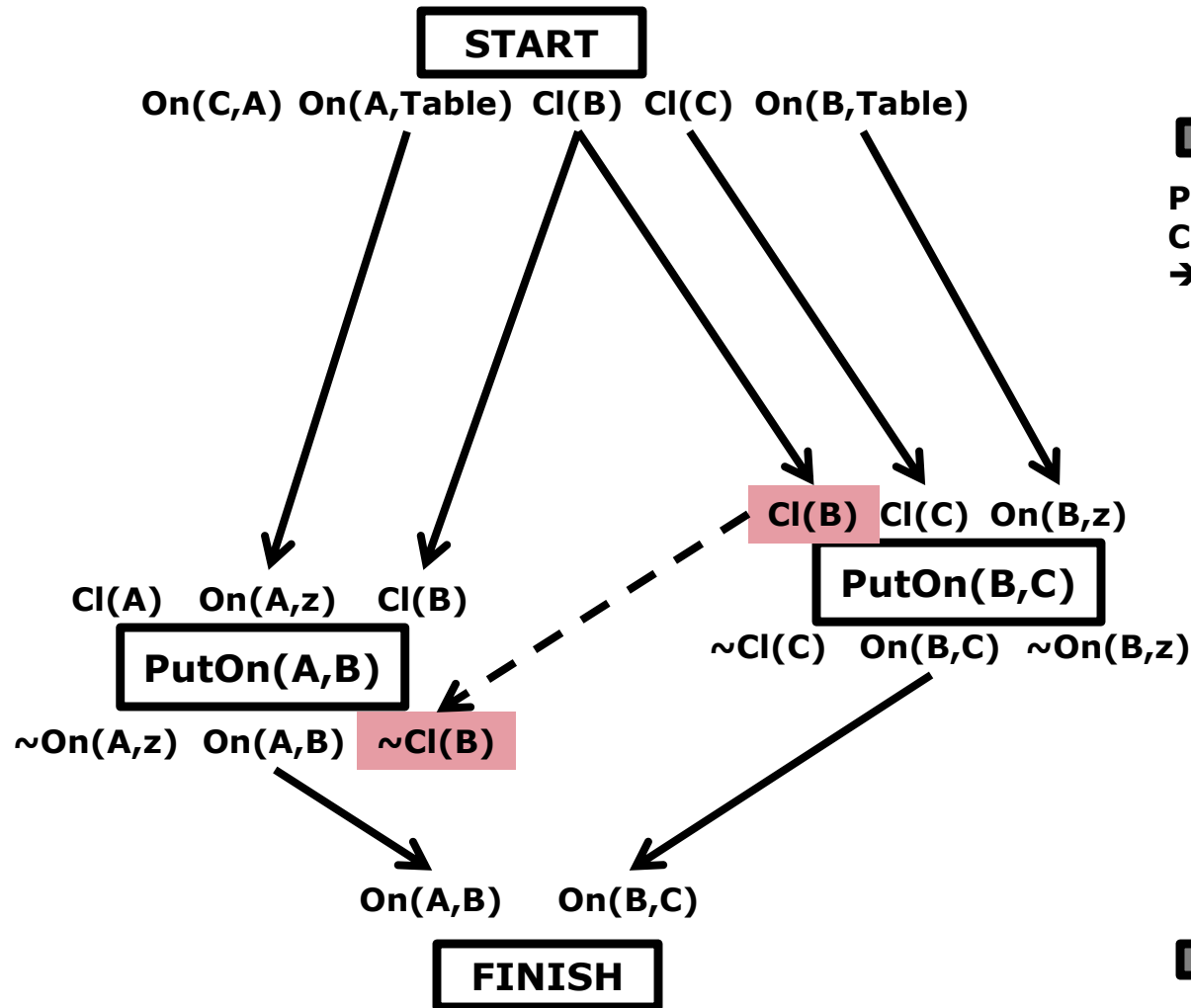
FINISH



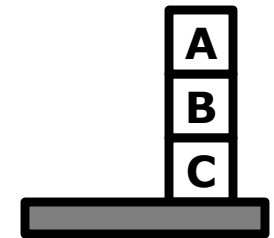
POP Solution... (cont.)



POP Solution... (cont.)



PutOn(A,B)
 Clobbers Cl(B)
 → Order after
 PutOn(B,C)



POP Solution... (cont.)

