Artificial Intelligence
Chapter 11: Planning

Michael Scherger
Department of Computer Science
Kent State University

Contents

- Search vs. Planning
- The Language of Planning Problems
- STRIPS Operators
- Partial Order Planning
- CLIPS

Introduction

- Planning is the task of coming up with a sequence of actions that will achieve the goal

- *Classical planning environments*
  - Fully observable, deterministic, finite, static (change only happens when the agent acts), and discrete (in time, action, objects)

Introduction

- A plan is ***complete*** iff every precondition is achieved
- A precondition is ***achieved*** iff it is the effect if an earlier step and no possibly intervening step undoes it

Search vs. Planning

- Consider a Internet Buying Agent whose task it is to buy our text book (search based)
  - ISBN# 0137903952
  - 10^9 possibilities
  - Searched based agent
    - One buying action for each ISBN number
    - Difficult to find a good heuristic
  - Planning based agent
    - Work backwards
    - Goal is *Have(0137903952)*
    - *Have(x)* results from *Buy(x)*
    - Therefore *Buy(0137903952)*

Search vs. Planning

- Now let's buy 4 books…
  - 10^{40} plans of just 4 steps using searching
    - Must search with a heuristic
      - Use # books remaining?
      - Not useful for our agent since it sees the goal test only as a black box that returns True or False for each state
      - Lacks autonomy; requires a human to supply a heuristic function for each new problem

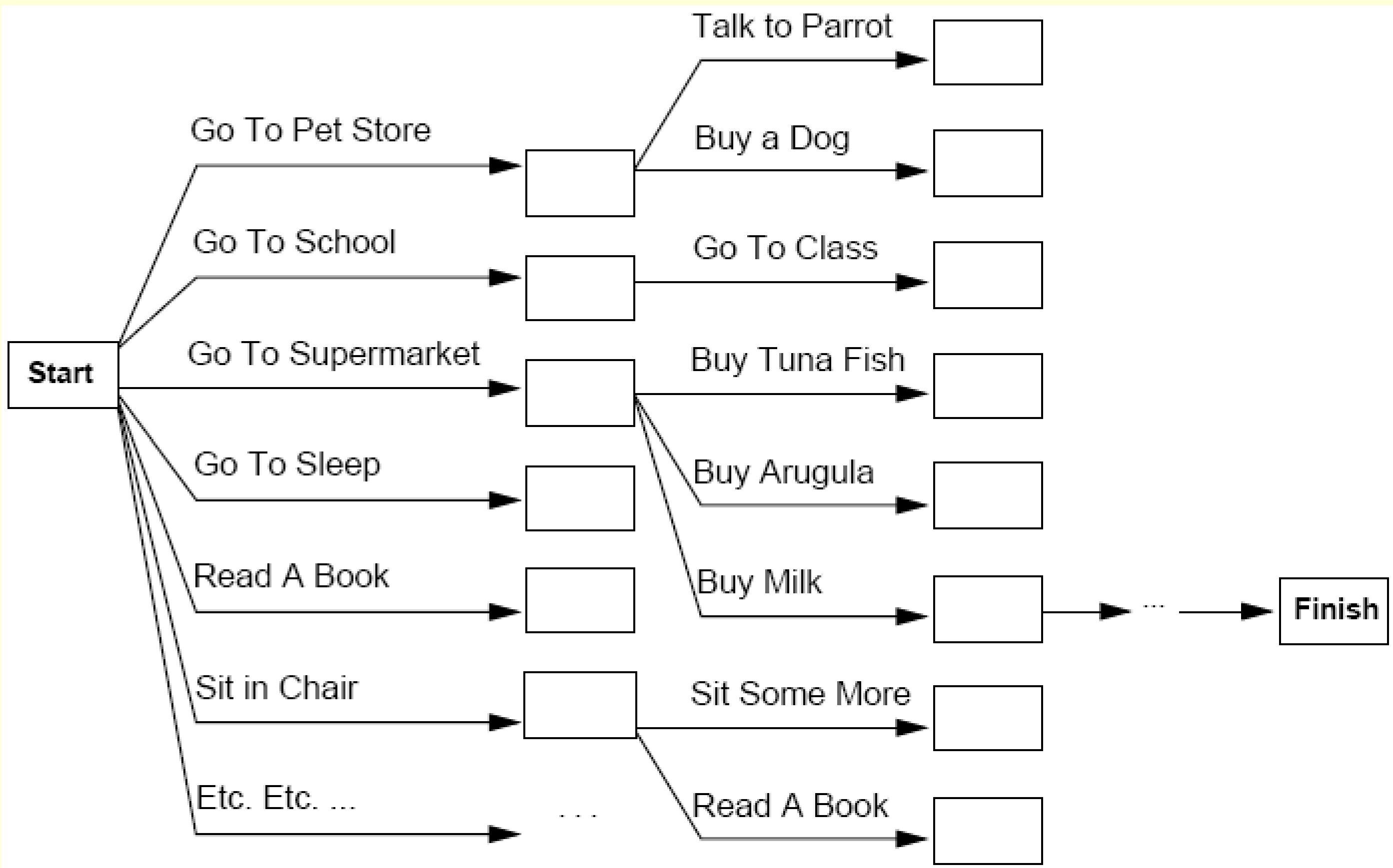
Search vs. Planning

- If our planning agent could use a conjunction of subgoals
  - Then it could use a single domain-independent heuristic
    - The number of unsatisfied conjuncts
  - Have(A) $\wedge$ Have(B) $\wedge$ Have(C) $\wedge$ Have(D)
  - A state containing Have(A) $\wedge$ Have(B) would have a cost 2

Search vs. Planning

- Another example:
  - Consider the task of getting milk, bananas, and a cordless drill
    - Really want to go to supermarket and then go to the hardware store
    - But we could get sidetracked!
      - By irrelevant actions

Search vs. Planning



Search vs. Planning

- Planning Systems do the following:
  - Open up action and goal representation to allow selection
  - Divide-and-conquer by sub-goaling
  - Relax requirement for sequential construction of solutions

The Language of Planning Problems

- Representation of states
  - Decompose the world into logical conditions and represent a state as a conjunction of positive literals
  - Must be ground and function-free
  - Examples:
    - Poor $\wedge$ Unknown
    - At($Plane_1$, CLE) $\wedge$ At($Plane_2$, LAS)
    - At(x,y) or At(Father(Fred), CLE) (not allowed)

The Language of Planning Problems

- Representation of goals
  - A partially specified state
  - Represented as a conjunction of ground literals
  - Examples
    - $At(\ Plane_1,\ LAS\ )$
    - $Rich \wedge Famous$
  - State ***s*** satisfies goal ***g*** if ***s*** contains all the atoms in ***g*** (and possibly others)
    - $Rich \wedge Famous \wedge Miserable$ satisfies $Rich \wedge Famous$

The Language of Planning Problems

- Representation of actions
  - Specified in terms of the preconditions that must hold before it can be executed and the effects that ensue when it is executed
  - Action(Fly(p, from, to))
    - Precond: At(p, from) $\wedge$ Plane(p) $\wedge$ Airport(from) $\wedge$ Airport(to)
    - Effect: $\neg$ At(p, from) $\wedge$ At(p, to)
  - This is also known as an ***action schema***

Search vs. Planning Again

- Search
  - States: program data structures
  - Actions: program code
  - Goal: program code
  - Plan: sequence from S_0

- Planning
  - States: logical sentences
  - Actions: preconditions and outcomes
  - Goal: logical sentences (conjunction)
  - Plan: constraints on actions

Example

- Suppose our current state is:
  - *At(P1, CLE)* $\wedge$ *At(P2, LAS)* $\wedge$ *Plane(P1)* $\wedge$ *Plane(P2)* $\wedge$ *Airport(CLE)* $\wedge$ *Airport(LAS)*

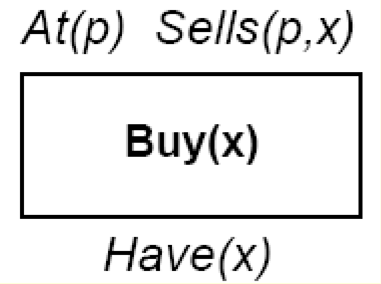
- This state satisfies the precondition
  - *At(p, from)* $\wedge$ *Plane(p)* $\wedge$ *Airport(from)* $\wedge$ *Airport(to)*

- Using the substitution
  - *{p/P1, from/CLE, to/LAS}*

- The following concrete action is applicable
  - *Fly(P1, CLE, LAS)*

STRIPS

- STanford Research Institute Problem Solver
- A restricted language for planning that describes actions and descriptions of objects in a system
- Example
  - Action: *Buy(x)*
  - Precondition: *At(p), Sells(p, x)*
  - Effect: *Have(x)*



STRIPS

- This abstracts away many important details!
- Restricted language -> 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

STRIPS

- Only positive literals in states:
  Poor $\wedge$ *Unknown*
- Closed world assumption:
  Unmentioned literals are false
- Effect P $\wedge$ $\neg$Q:
  Add P and delete Q
- Only ground literals in goals:
  Rich $\wedge$ *Famous*

STRIPS

- Goals are conjunctions:
  $Rich \wedge Famous$
- Effects are conjunctions:
- No support for equality
- No support for types

ADL

- Positive and negative literals in states:
  $\neg$ Rich $\wedge$ $\neg$ Famous

- Open world assumption:
  Unmentioned literals are unknown

- Effect $P \wedge \neg Q$:
  Add P and $\neg$ Q and delete $\neg$ P and Q

- Quantified variables in goals:
  $\exists x\ At(P_1, x) \wedge At(P_2, x)$ is the goal of having P_1 and P_2 in the same place

ADL

- Goals allow conjunction and disjunction:
  $\neg$ *Poor* $\wedge$ *(Famous* $\vee$ *Smart)*

- Conditional Effects are allowed:
  when *P: E means E is an effect only if P is satisfied*

- Equality predicate built in:
  (x = y)

- Variables can have types:
  (p : Plane)

Example: Air Cargo Transport

- Init($At(C_1, CLE) \wedge At(C_2, LAS) \wedge At(P_1, CLE) \wedge At(P_2, LAS) \wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2) \wedge Airport(CLE) \wedge Airport(LAS)$)

- Goal($At(C_1, LAS)$ $At(C_2, CLE)$)

Example: Air Cargo Transport

- *Action(Load(c, p, a),*
  Precond: At(c, a) $\wedge$ At(p, a) $\wedge$ Cargo(c) $\wedge$ Plane(p) $\wedge$ Airport(a)
  Effect: $\neg$ At(c, a) $\wedge$ In(c, p))

- *Action(Unload(c, p, a),*
  Precond: In(c, p) $\wedge$ At(p, a) $\wedge$ Cargo(c) $\wedge$Plane(p) $\wedge$ Airport(a)
  Effect: At(c, a) $\wedge$ $\neg$ In(c, p))

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

Example: Air Cargo Transport

[Load(C_1, P_1, CLE), Fly(P_1, CLE, LAS),
Unload(C_1, P_1, LAS),
Load(C_2, P_2, LAS), Fly(P_2, LAS, CLE),
Unload(C_2, P_2, CLE)]

- Is it possible for a plane to fly to and from the same airport?

Example: The Spare Tire Problem

- Init(At(Flat, Axle) $\wedge$ At(Spare, Trunk))
- Goal(At(Spare, Axle))

Example: The Spare Tire Problem

- *Action(Remove(Spare, Trunk),*
  Precond: At(Spare, Trunk)
  Effect: ¬ At(Spare, Trunk) ∧ At(Spare, Ground))

- *Action(Remove(Flat, Axle),*
  Precond: At(Flat, Axle)
  Effect: ¬ At(Flat, Axle) ∧ At(Flat, Ground))

- *Action(PutOn(Spare, Axle),*
  Precond: At(Spare, Ground) ∧ ¬ At (Flat, Axle)
  Effect: ¬ At(Spare, Ground) ∧ At(Spare, Axle))

- *Action(LeaveOvernight,*
  Precond:
  Effect: ¬ At(Spare, Ground) ∧ ¬ At(Spare, Axle) ∧ ¬ At(Spare, Trunk) ∧ ¬ At(Flat, Ground) ∧ ¬ At(Flat, Axle)

Example: The Blocks World

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

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

Example: The Blocks World

- *Action(Move(b, x, y),*
  Precond: $On(b,x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge (b \neq x) \wedge (b \neq y) \wedge (x \neq y)$
  Effect: $On(b, y) \wedge Clear(x) \wedge \neg On(b,x) \wedge \neg Clear(y))$

- *Action(MoveToTable(b, x),*
  Precond: $On(b, x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x)$
  Effect: $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$

Example: The Blocks World

- A plan for building a three block tower

- [Move(B, Table, C), Move(A, Table, B)]

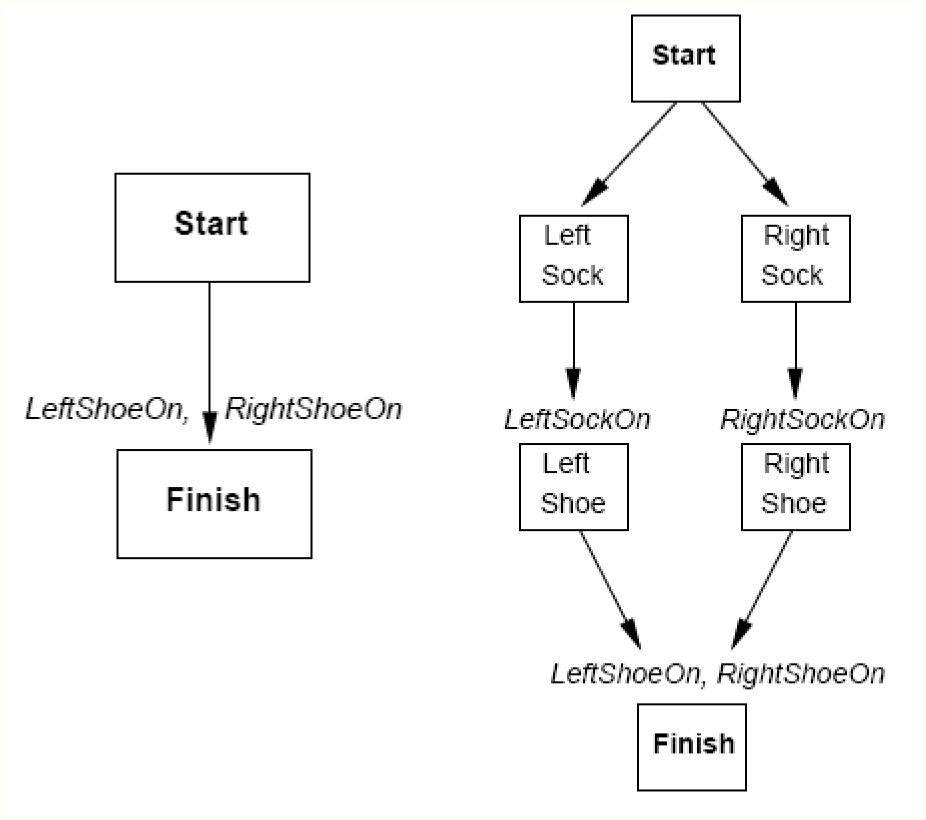
Partially Ordered Plans

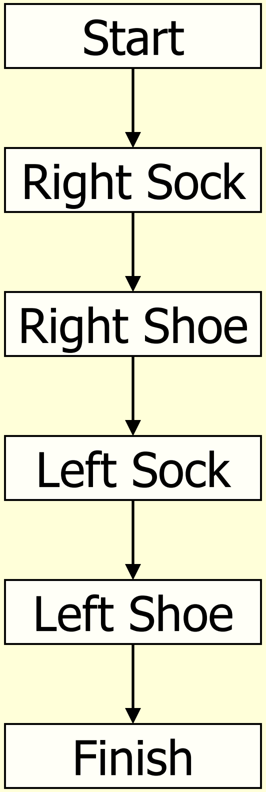
- Partially Ordered Plan
  - A partially ordered collection of steps
    - ***Start step*** has the initial state description and its effect
    - ***Finish step*** has the goal description as its precondition
    - ***Causal links*** from outcome of one step to precondition of another step
    - ***Temporal ordering*** between pairs of steps

Partial Ordered Plans

- An open condition is a 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 if an earlier step and no possibly intervening step undoes it

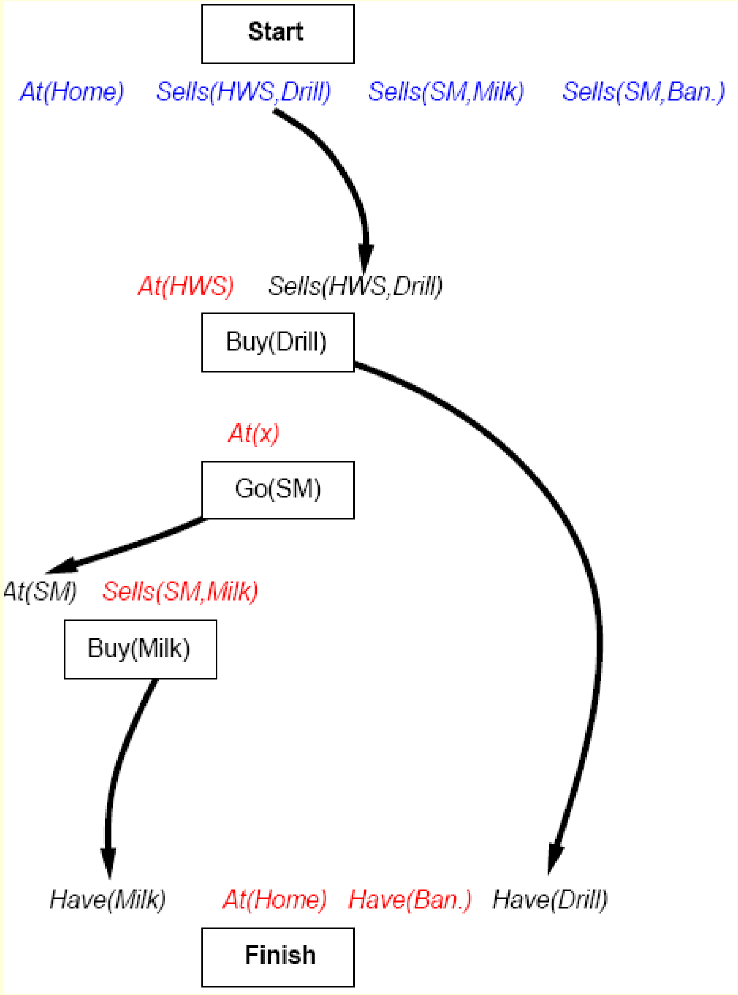
Partially Ordered Plans



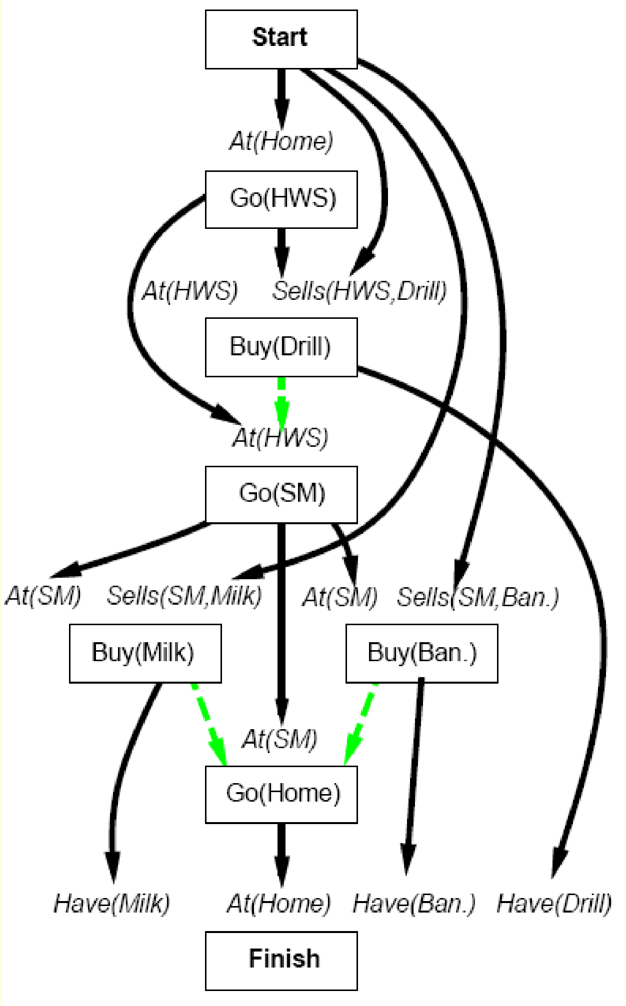


Partially Ordered Plans

Partially Ordered Plans



Partially Ordered Plans



POP Algorithm

function POP(*initial, goal, operators*) **returns** *plan*

$plan \leftarrow$ MAKE-MINIMAL-PLAN(*initial, goal*)
loop do
  if SOLUTION?(*plan*) **then return** *plan*
  $S_{need},\ c \leftarrow$ 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

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*)

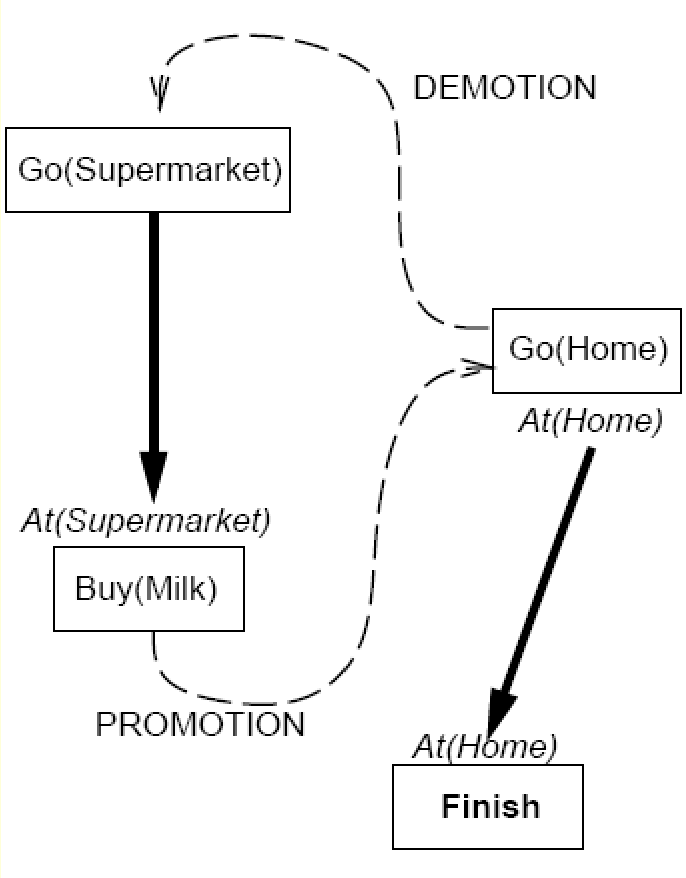
---

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

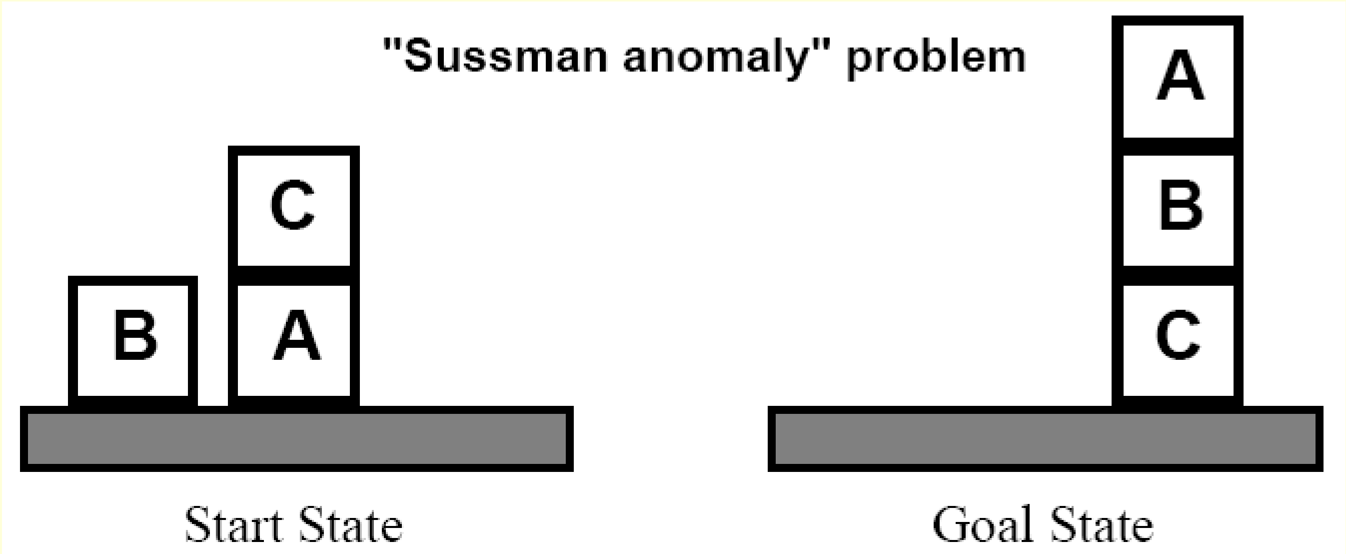
Clobbering

- A ***clobberer*** is a potentially intervening step that destroys the condition achieved by a causal link
  - Example Go(Home) clobbers At(Supermarket)



- Demotion
  - Put before Go(Supermarket)
- Promotion
  - Put after Buy(Milk)

Example: Blocks World



Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y) Clear(z) On(x,y)

Clear(x) On(x,z)

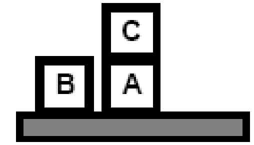
PutOnTable(x)

~On(x,z) Clear(z) On(x,Table)

Example: Blocks World

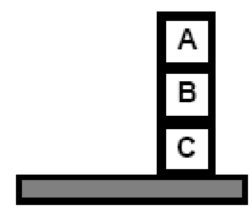
START

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

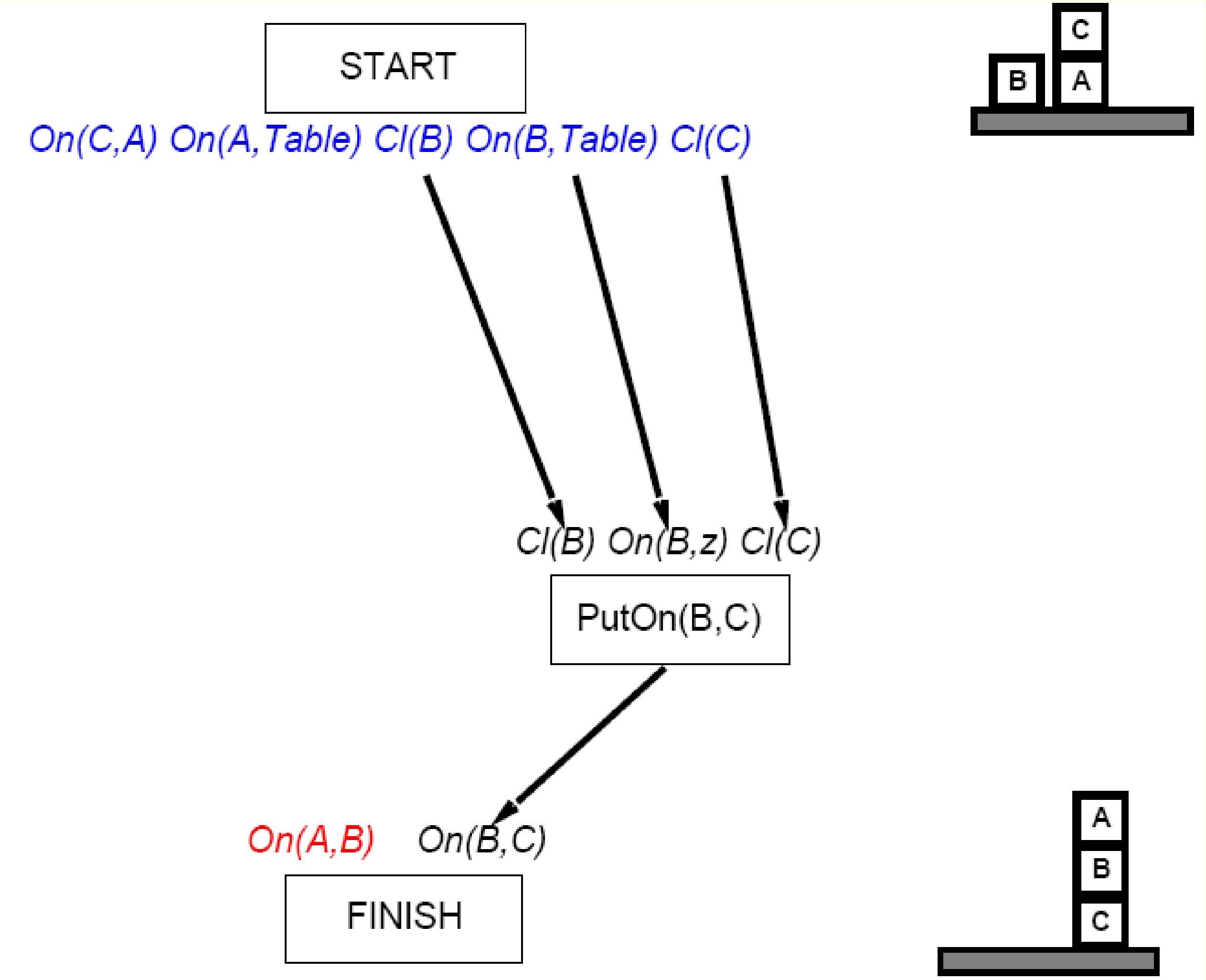


On(A,B) *On(B,C)*

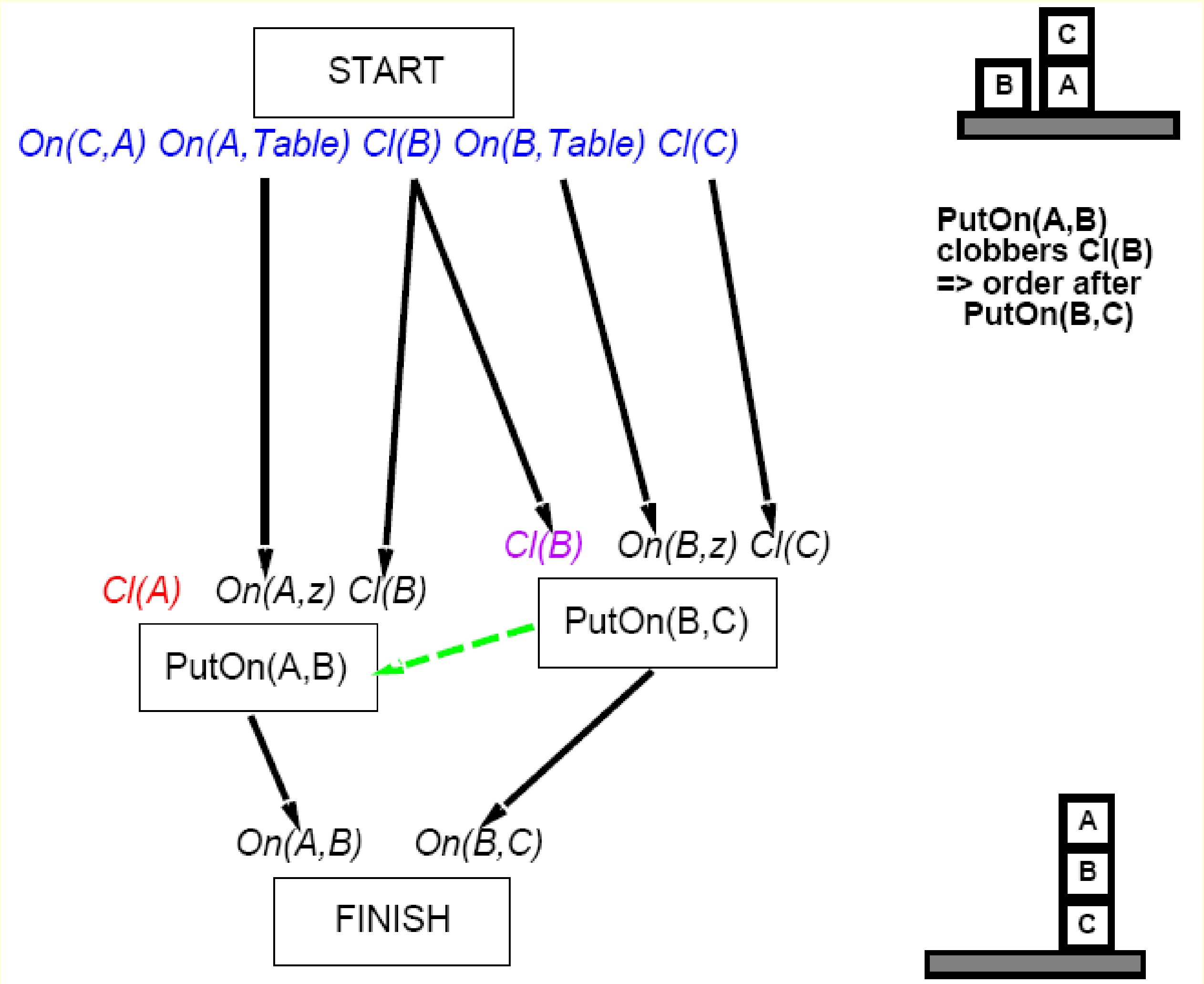
FINISH



Example: Blocks World



Example: Blocks World



Example: Blocks World