Principles of Intelligent Systems: Planning with STRIPS*

Lecture 12

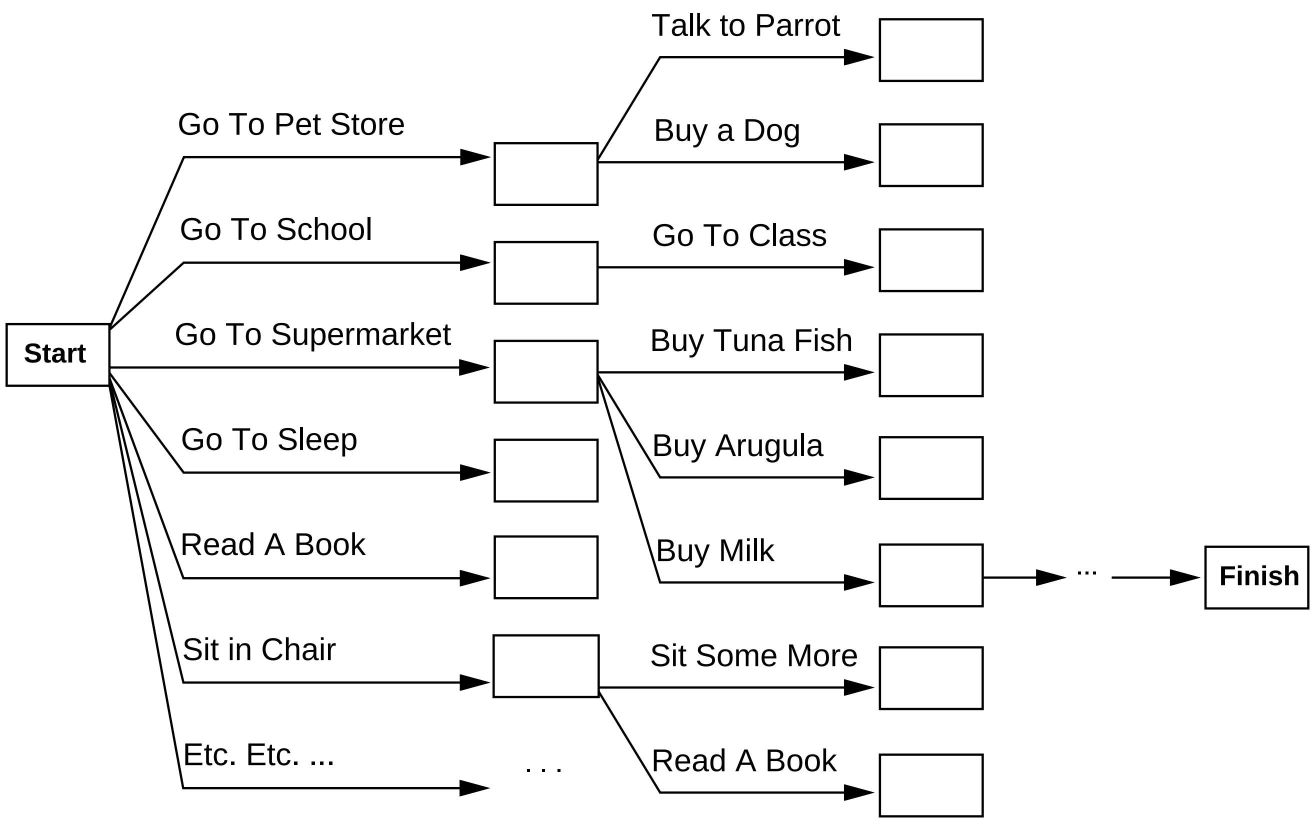
*These slides are based on the Chapter 11 slides of Russell and Norvig's *Artificial Intelligence: A modern approach* (http://aima.eecs.berkeley.edu/slides-pdf/)

Outline

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

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 contd.

Planning systems do the following:

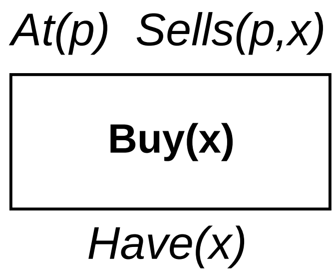
1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

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

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

Also requires the specification of an initial state and a goal state

A complete set of STRIPS operators can be translated into a set of successor-state axioms, i.e. into Situation Calculus

Expressiveness of STRIPS

STRIPS (STanford Research Institute Problem Solver) provides a "cut-down" first-order logic representation for planning:

- Preconditions and effects must be function-free
- Closed world assumption: Unmentioned literals are false
- Effect $P \wedge \neg Q$: add P and delete Q
- Only positive literals in states e.g.: $Poor \wedge Unknown$
- Only ground literals in goals e.g.: $Rich \wedge Famous$
- Goals and effects are conjunctions e.g.: $Rich \wedge Famous$
- No support for equality e.g. $x = y$ is not allowed

Note, the closed-world assumption avoids the frame problem

As an example, consider the following air transport problem involving loading and unloading cargo onto and off planes and flying it from place to place:

Example STRIPS problem

$Init(At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK)$
$\wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2)$
$\wedge Airport(JFK) \wedge Airport(SFO))$
$Goal(At(C_1, JFK) \wedge At(C_2, SFO))$
$Action(Load(c, p, a),$
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: $\neg At(p, from) \wedge At(p, to))$

Planning as state-space search

Forward state-space search: As each action has an effect, planning can be solved using state-space search algorithms

This requires the following components:

An initial state: given in the STRIPS definition
Actions applicable for any given state that specify the successor state:
given by the STRIPS action effects
A goal test: given in the STRIPS definition
A step cost: each action is given a cost of one

Similarly, as each action has a precondition, we can search backwards from the goal using backward state-space search

In either case we can use heuristics to estimate the cost of reaching the goal and apply algorithms like A^*

Partially ordered plans

However, we can also attempt to find a plan by solving several sub-problems *simultaneously* and combining them - this can have the advantage of reducing the size of the search space and providing a more flexible answer:

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

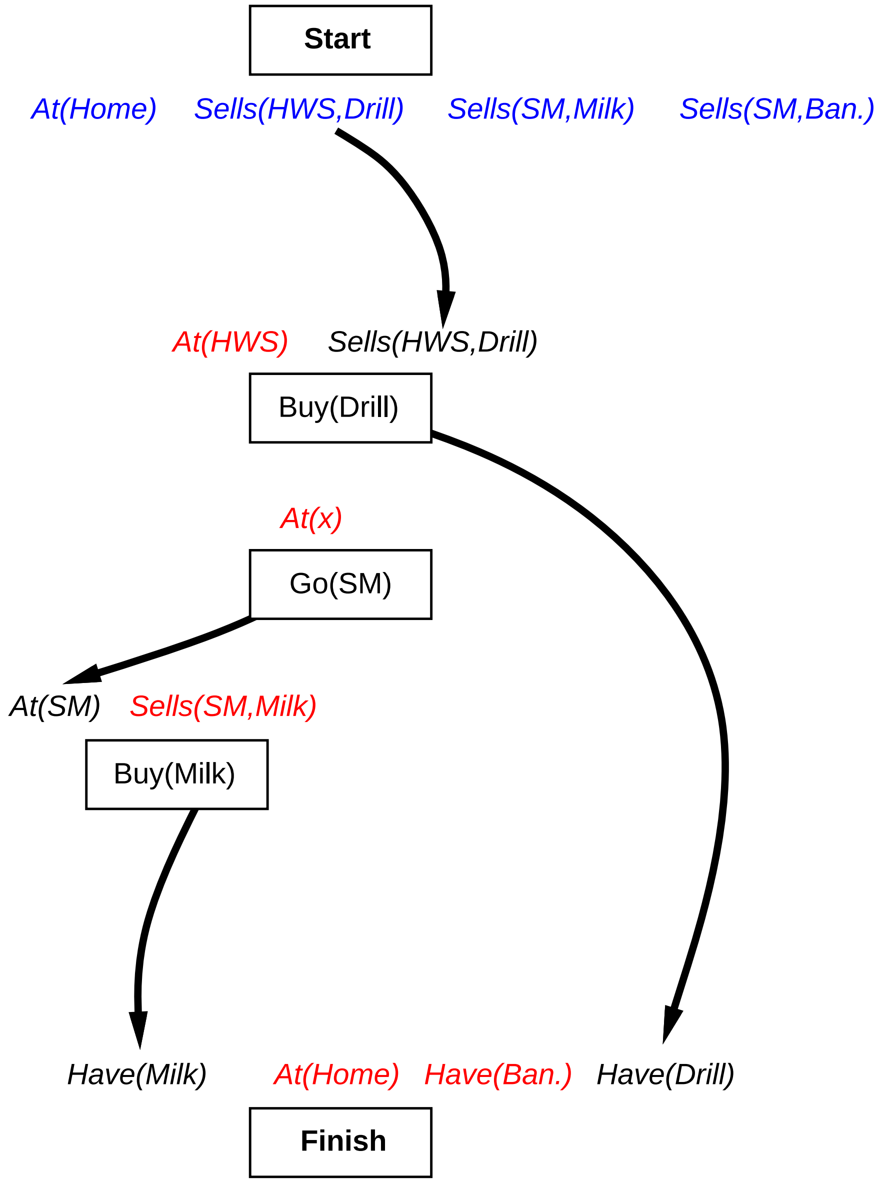
A plan is complete iff every precondition is achieved

A precondition is achieved iff it is the effect of an earlier step
and no possibly intervening step undoes it

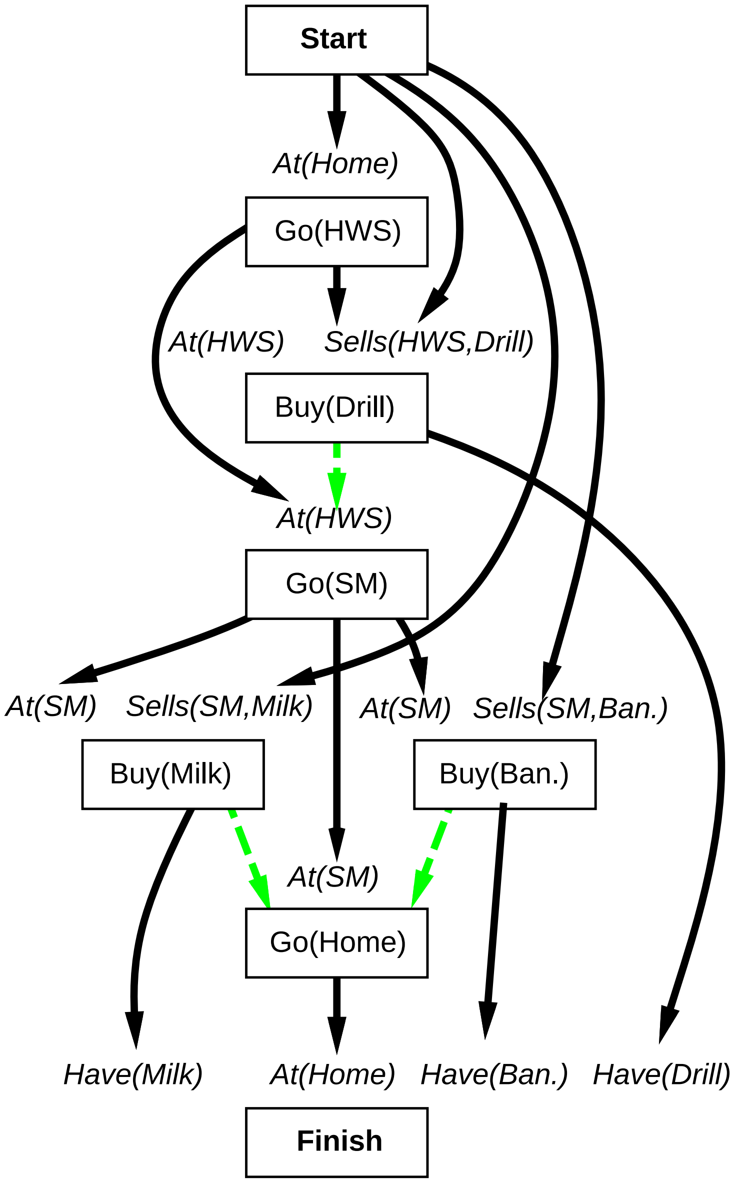
Example


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

Example



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

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

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 contd.

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*)

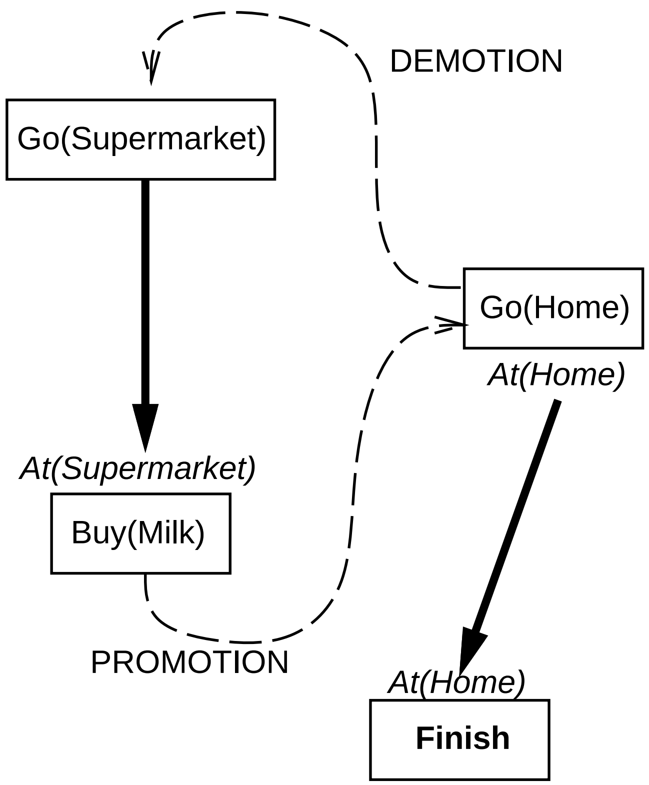
---

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 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)

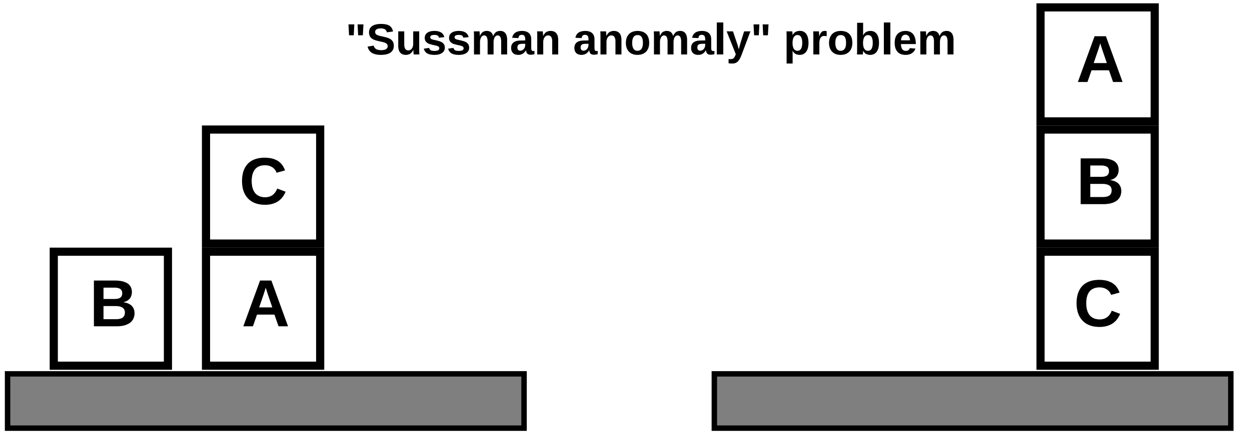
Extensions for disjunction, universals, negation, conditionals

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

Goal State

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)

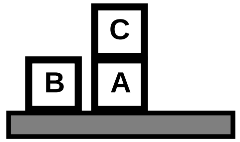
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

Example contd.

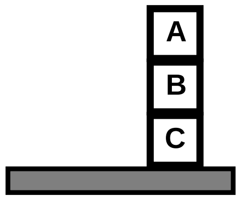
START

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

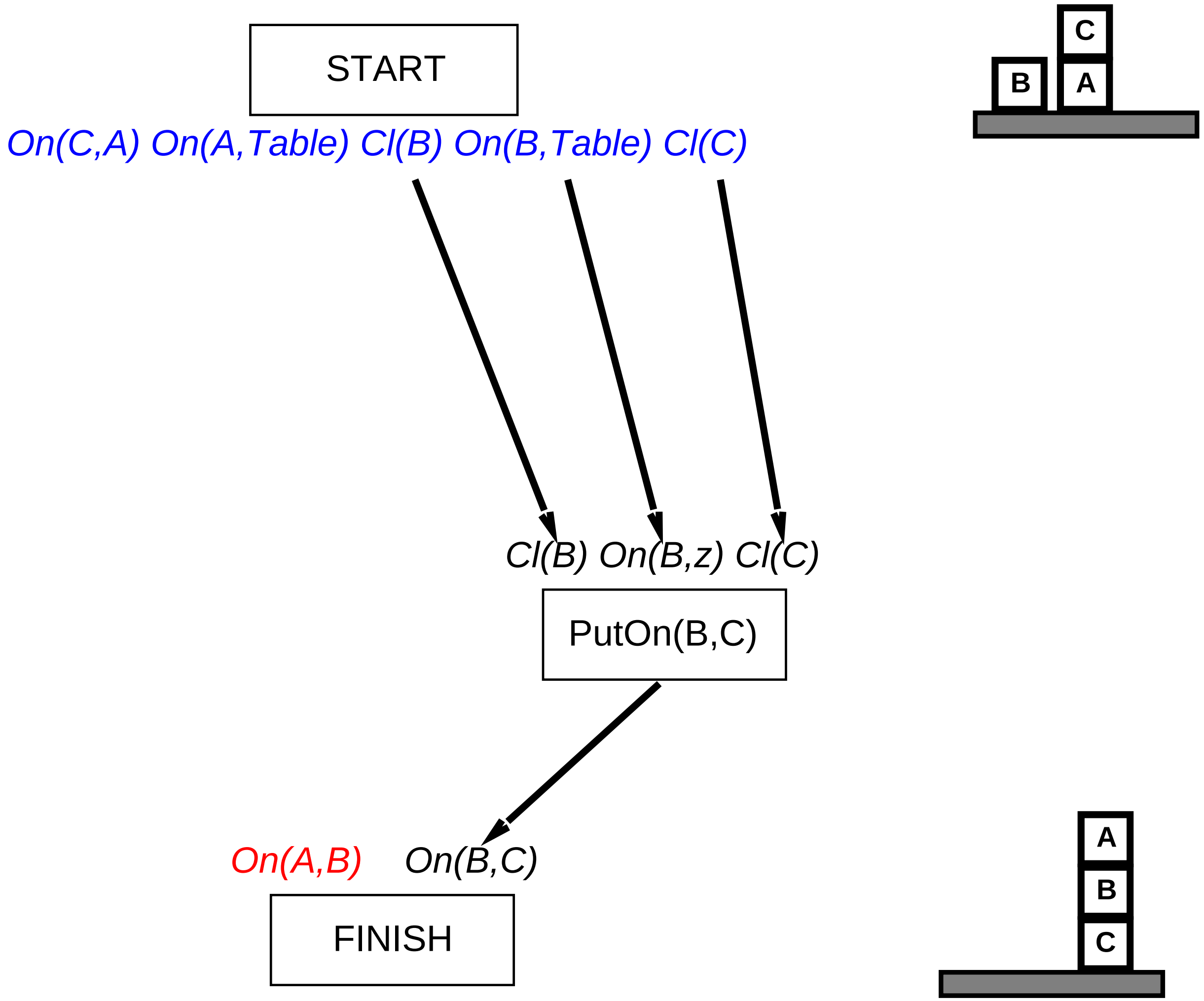


On(A,B) *On(B,C)*

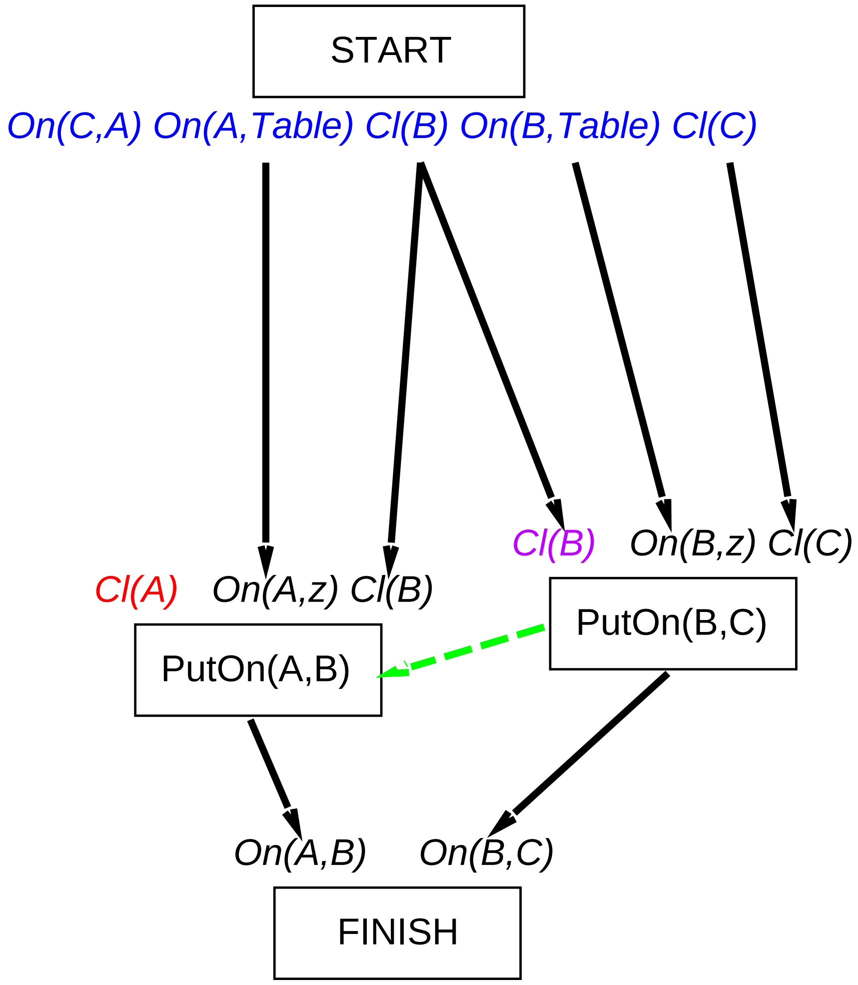
FINISH

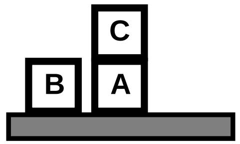


Example contd.

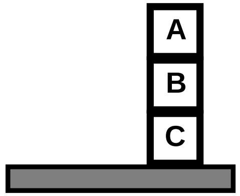


Example contd.

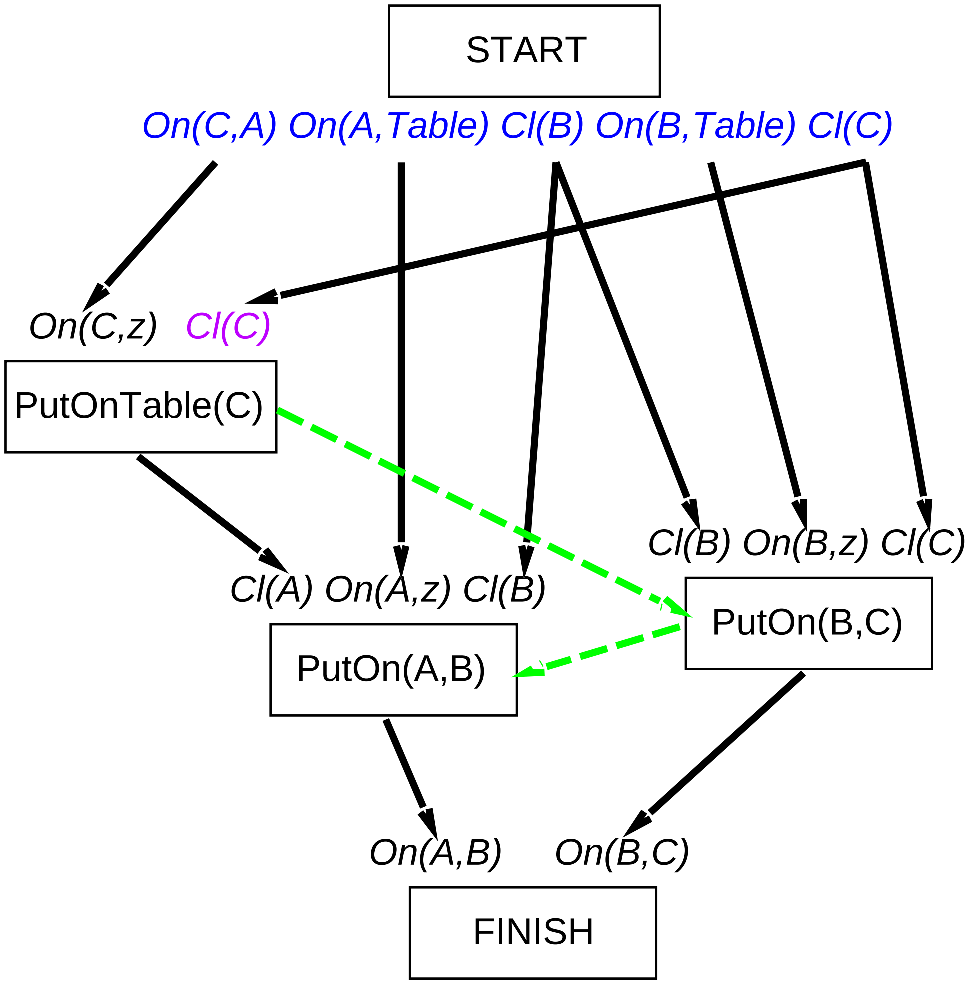


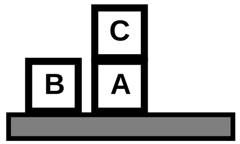


PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)



Example contd.





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

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