Artificial Intelligence

Topic 9

Planning

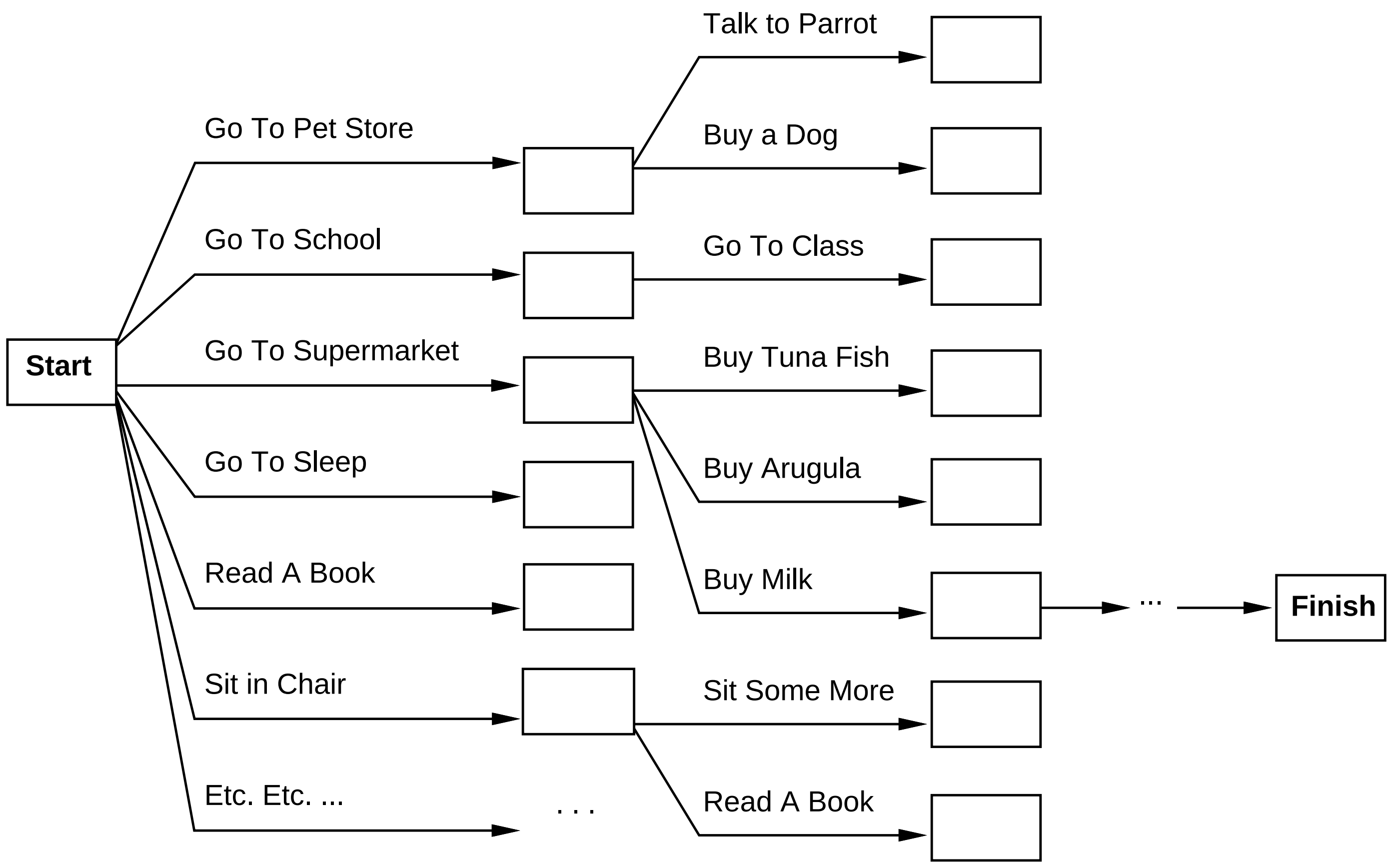
- ◇ Search vs. planning
- ◇ Planning Languages and STRIPS
- ◇ State Space vs. Plan Space
- ◇ Partial-order Planning

Reading: Russell & Norvig, Chapter 11

1. 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

1. 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	internal state of Java objects	descriptive (logical) sentences
Actions	encoded in Java methods	preconditions/outcomes
Goal	encoded in Java methods	descriptive sentence
Plan	sequence from s_0	constraints on actions
	$\Rightarrow$ *implicit*	$\Rightarrow$ *explicit*
	$\Rightarrow$ *hard to decompose*	$\Rightarrow$ *easier to decompose*

2. Planning Languages and STRIPS

Require *declarative language* — *declarations* or *statements* about world.

Range of logics have been proposed — best descriptive languages we have, but can be difficult to use in practice.

more descriptive power $\longrightarrow$ *more difficult to compute (reason) automatically*

STRIPS (STanford Research Institute Problem Solver) first to suggest suitable compromise

- restricted form of logic
- restricted language $\Rightarrow$ efficient algorithm

Basis of many subsequent languages and planners.

States

At(Home), $\neg$ *Have(Milk),* $\neg$ *Have(Bananas),* $\neg$ *Have(Drill)*

(conjunctions of function-free ground literals)

2. Planning Languages and STRIPS

Goals

At(Home), Have(Milk), Have(Bananas), Have(Drill)

Can have variables

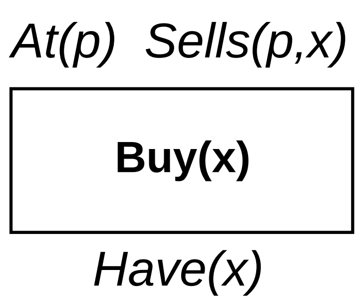
At(x), Sells(x,Milk)

(conjunctions of function-free literals)

Actions

Action (Name): $Buy(x)$
Precondition: $At(p), Sells(p,x)$
Effect: $Have(x)$

(Precondition: conjunction of positive literals
Effect: conjunction of literals)



3. State Space vs. Plan Space

Standard search: node = concrete world state
Planning search: node = *partial plan*

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

Operators on partial plans, eg:

- add a step to fulfill an open condition
- order one step wrt another
- instantiate an unbound variable

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

4. Partial-order planning

Example

Goal: *RightShoeOn, LeftShoeOn*

Operators:

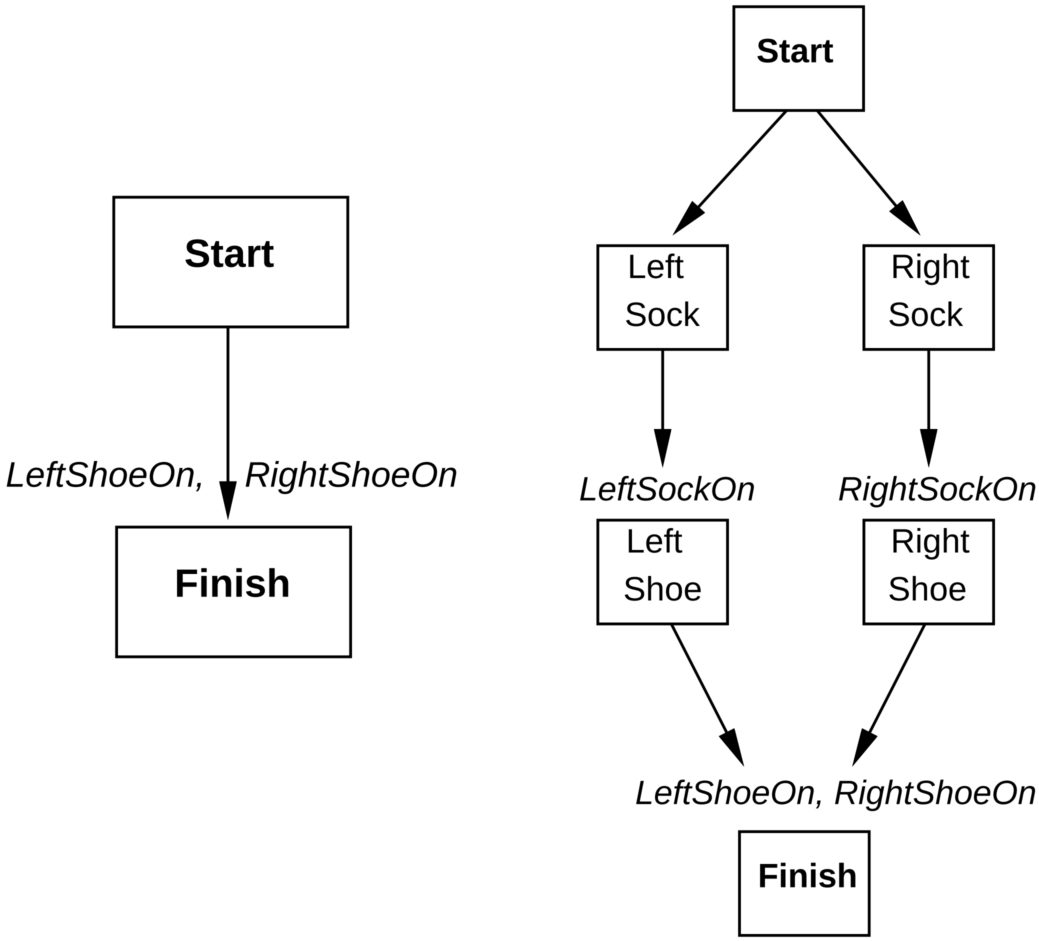
Op(Action: *RightShoe*, Precond: *RightSockOn*, Effect: *RightShoeOn*)
Op(Action: *RightSock*, Effect: *RightSockOn*)
Op(Action: *LeftShoe*, Precond: *LeftSockOn*, Effect: *LeftShoeOn*)
Op(Action: *LeftShoe*, Effect: *LeftShoeOn*)

Consider partial plans:

1. *LeftShoe, RightShoe* — ordering unimportant
2. *RightSock, RightShoe* — ordering important
3. *RightSock, LeftShoe, RightShoe* — ordering between *some* actions important

partial order planner ⇒ planner that can represent steps in which some are ordered (in sequence) and others not (in "parallel")

4. Partial-order planning



least commitment planner — partial order planner that *delays commitment to order between steps for as long as possible*

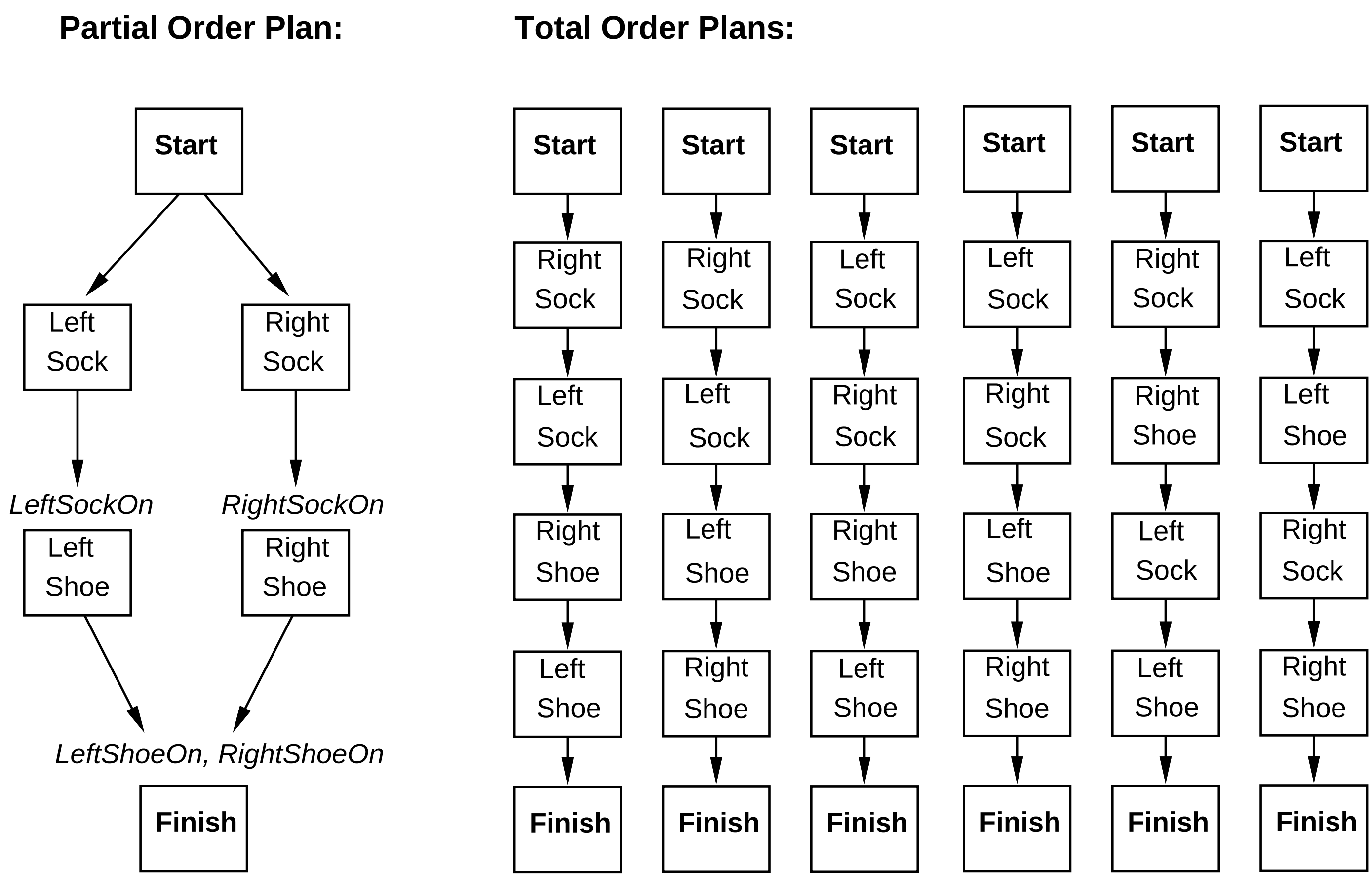
⇒ less backtracking

A plan is <u>complete</u> iff every precondition is achieved

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

4. Partial-order planning

linearisation — obtaining a totally ordered plan from a partially ordered plan by imposing ordering constraints



4. Partial-order planning

In addition to orderings we must record

- variable bindings: eg. $x = LocalStore$
- causal links: $S_i \xrightarrow{c} S_j$ (S_i achieves precondition c for S_j)

Thus our initial plan might be:

Plan(STEPS:{ S_1: *Op*(Action: *Start*),
S_2: *Op*(Action: *Finish*,
Precond: *RightShoeOn*, *LeftShoeOn*)},
ORDERINGS: { $S_1 \prec S_2$ },
BINDINGS: {},
LINKS: {})

Algorithm $\cdots \rightsquigarrow$

4.1 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

continued...

4.1 POP algorithm sketch

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*)

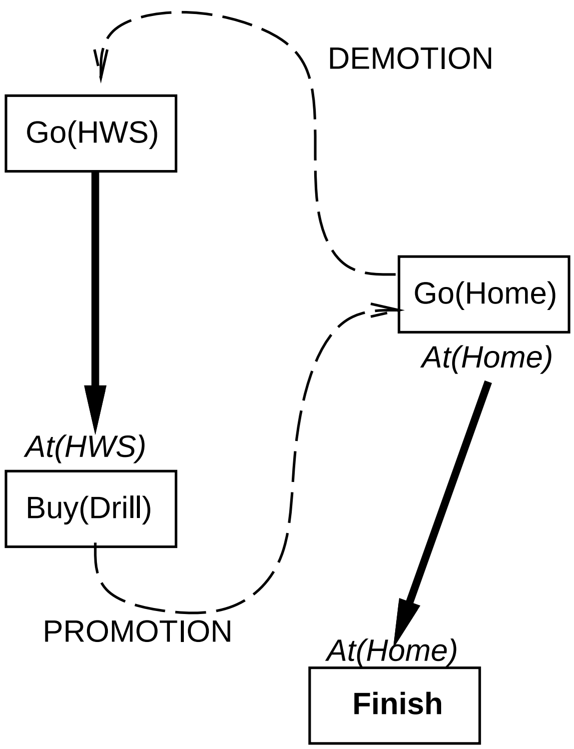
for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
choose either
Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
if not CONSISTENT(*plan*) **then fail**
end

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

Extensions for more expressive languages (eg disjunction, etc)

4.2 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)*:

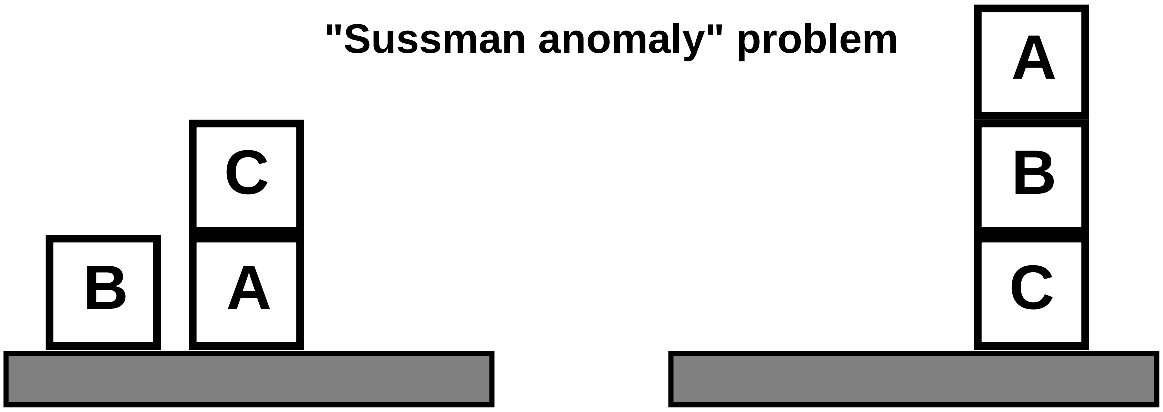


Demotion: put before $Go(HWS)$

Promotion: put after $Buy(Drill)$

4.3 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)

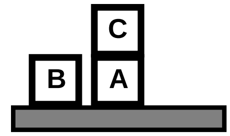
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

4.3 Example: Blocks world

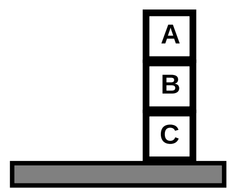
START

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

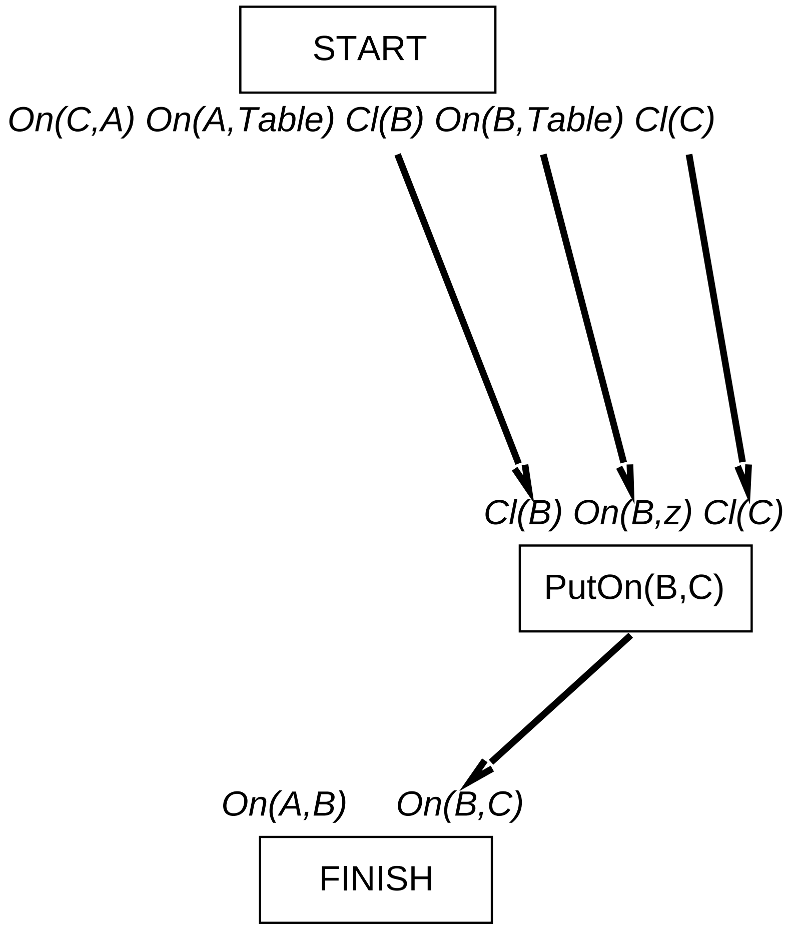


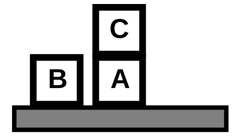
On(A,B) *On(B,C)*

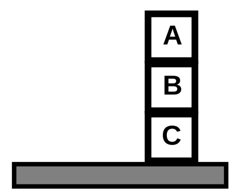
FINISH



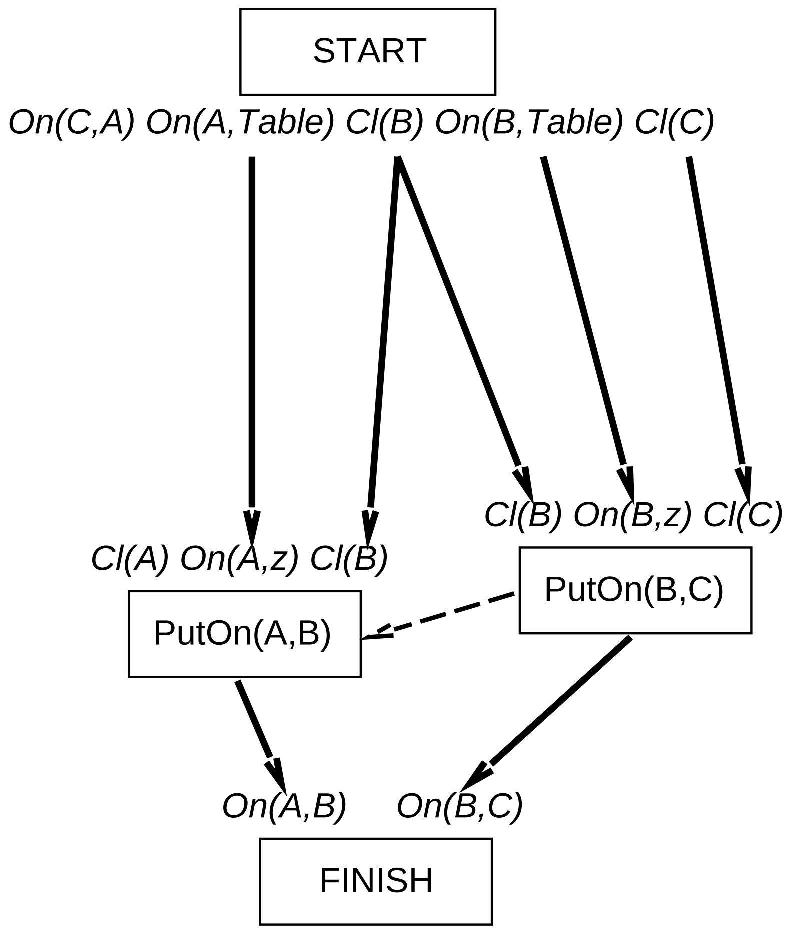
4.3 Example: Blocks world

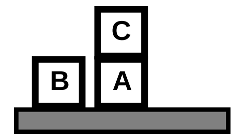




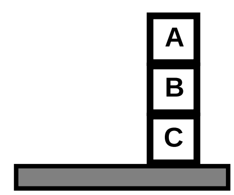


4.3 Example: Blocks world

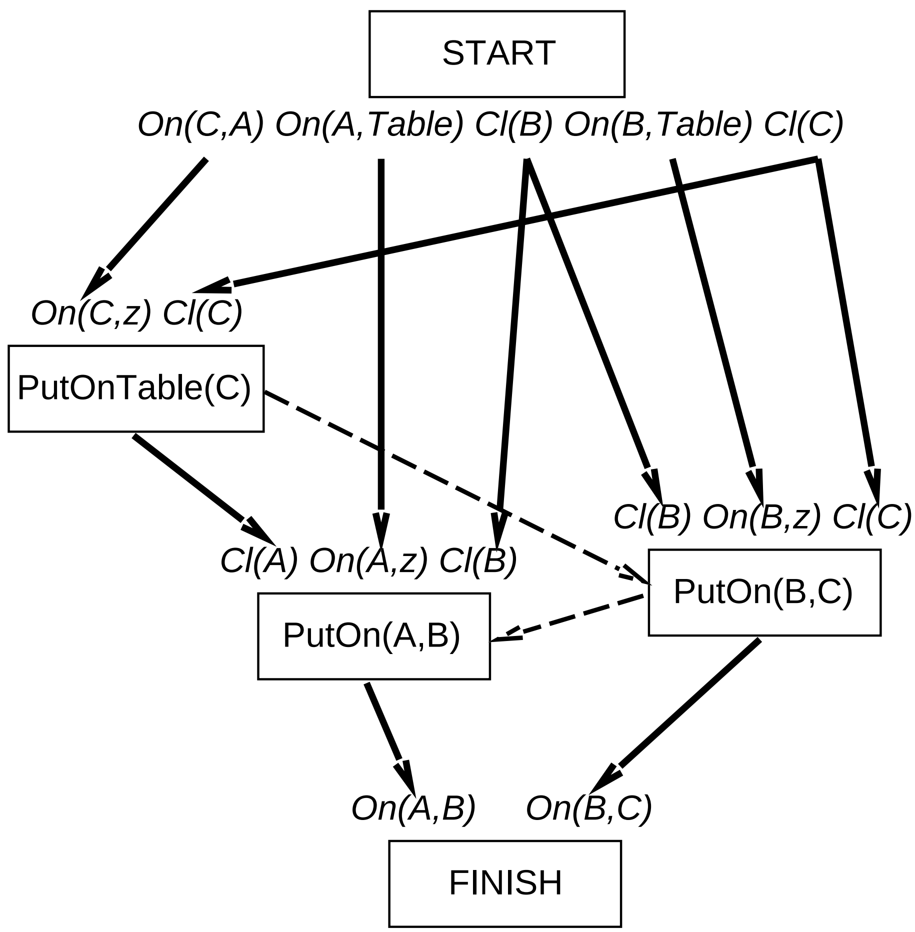


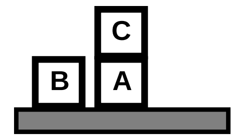


PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)



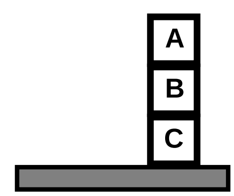
4.3 Example: Blocks world





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

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



The End