07-380 AI & ML II

Classical Planning
or Symbolic Planning

Instructors: Pat Virtue & Mohammad Salameh

Slide credits: CMU AI

Today

Polls covering Pre-reading

- Planning with Propositional Logic
- From Propositional Logic to Classical Planning
- Classical Planning Representation
- Solving Planning Problems: State space search

New

- Solving Planning Problems: Planning Graphs and GraphPlan

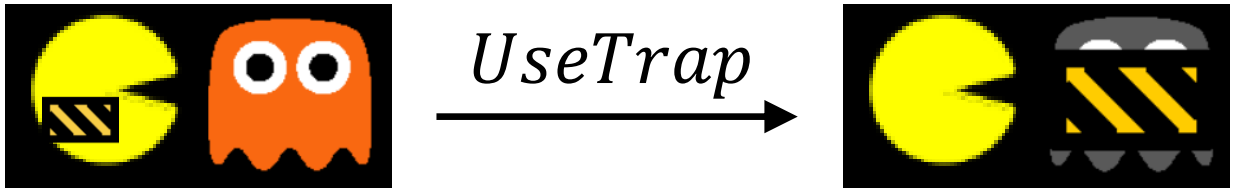
Next time

- From GraphPlan to Modern Planning
- Planning with Continuous States

Pre-reading

Planning with Propositional Logic

Poll 1

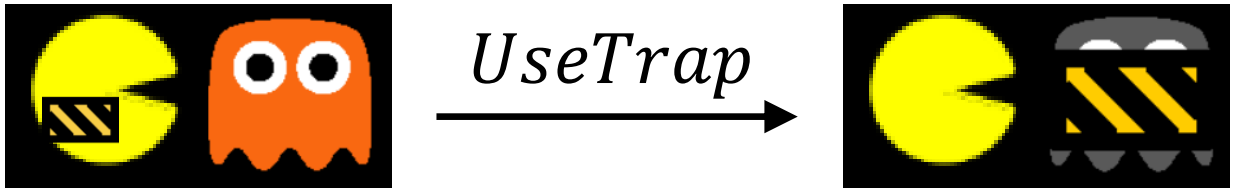


Which models satisfy the following sentence?

$$\alpha\text{: } HaveTrap \wedge UseTrap \Rightarrow \neg HaveTrap$$

	$HaveTrap$	$UseTrap$
A.	True	True
B.	True	False
C.	False	True
D.	False	False

Poll 1



Which models satisfy the following sentence?

$$\alpha\text{: } HaveTrap \wedge UseTrap \Rightarrow \neg HaveTrap$$

	$HaveTrap$	$UseTrap$	$HaveTrap \wedge UseTrap$	$\neg HaveTrap$	α
A.	True	True	True	False	False
B.	True	False	False	False	True
C.	False	True	False	True	True
D.	False	False	False	True	True

Fluents & Effect Axioms

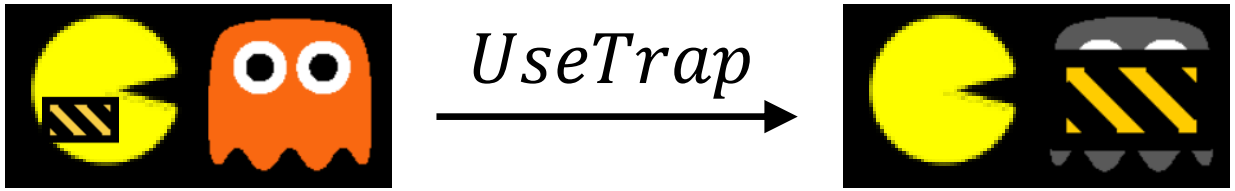
$\xrightarrow{UseTrap}$

Effect axioms are super fragile

$$\beta\text{: } HaveTrap_t \wedge UseTrap_t \Rightarrow \neg HaveTrap_{t+1}$$

$HaveTrap_t$	$UseTrap_t$	$HaveTrap_{t+1}$	LHS	β	
True	True	True	True	False	N/A (β is not satisfied)
True	False	True	False	True	Correct
False	True	True	False	True	Illegal and magic
False	False	True	False	True	Magic (trap appears!)
True	True	False	True	True	Correct
True	False	False	False	True	Lost trap?
False	True	False	False	True	Illegal (trap without a trap)
False	False	False	False	True	Correct

Successor State Axioms



More robust solution to frame problem

and nothing caused it to change

$$\gamma: \neg HaveTrap_{t+1} \Leftrightarrow [\neg HaveTrap_t] \vee [HaveTrap_t \wedge UseTrap_t]$$

Planning with Propositional Logic

The Frame Problem: *how do we deal with time?*

Fluents:

- Any symbols that may need to change over time: $HaveTrap_{t=1}$ and $HaveTrap_{t=2}$
- Number of symbols scales with T, number of models scales with 2^T

Effect Axioms:

- Super fragile, even with proper fluents
- $PRECONDITION_t \wedge ACTION_t \Rightarrow EFFECT_{t+1}$

One solution: Successor state axioms

- More robust, but verbose and difficult to read/write
- For fluent X_t:

$$X_t \Leftrightarrow [\ X_{t-1} \wedge \neg(\text{some action}_{t-1} \text{ made it false})]\ \vee$$
$$[\neg X_{t-1} \wedge (\text{some action}_{t-1} \text{ made it true})]$$

Planning as Satisfiability: SATPlan

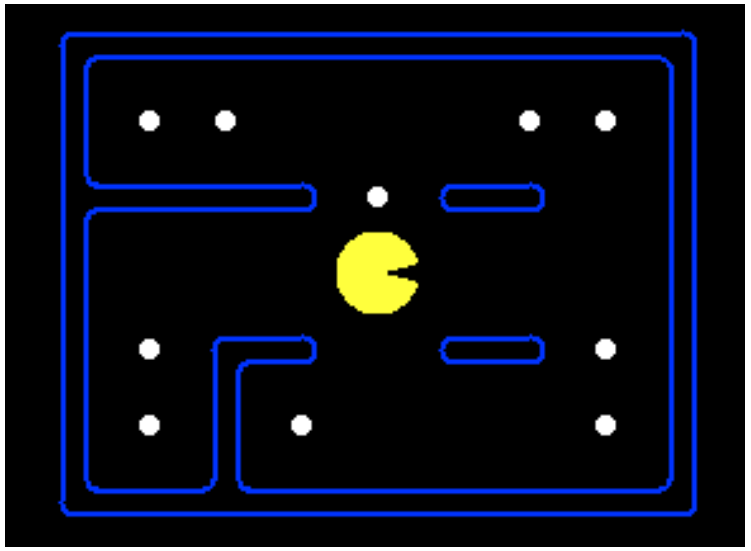
Given a hyper-efficient SAT solver, can we use it to make plans?

Yes, for fully observable, deterministic case: planning problem is solvable iff there is some satisfying assignment for actions etc.

For T = 1 to infinity, set up the KB as follows and run SAT solver:

- Initial state, domain constraints
- For t=1 to T
  - Successor state axioms for time t
  - Precondition axioms: $At_1,1_0 \wedge N_0 \Rightarrow \neg Wall_1,2$ etc.
  - Action exclusion axioms: $\neg(N_0 \wedge W_0) \wedge \neg(N_0 \wedge S_0) \wedge ..$ etc.
- Goal is true at time T

Planning as Satisfiability: SATPlan



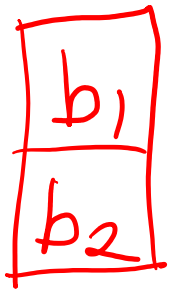
```
(P[4,3,0] & (~P[1,1,0] | ~P[1,2,0]) & (~P[1,1,0] | ~P[1,3,0]) & (~P[1,1,0] | ~P[1,5,0]) & (~P[1,1,0] |
~P[2,3,0]) & (~P[1,1,0] | ~P[2,5,0]) & (~P[1,1,0] | ~P[3,1,0]) & (~P[1,1,0] | ~P[3,3,0]) & (~P[1,1,0] |
~P[3,5,0]) & (~P[1,1,0] | ~P[4,1,0]) & (~P[1,1,0] | ~P[4,2,0]) & (~P[1,1,0] | ~P[4,3,0]) & (~P[1,1,0] |
~P[4,4,0]) & (~P[1,1,0] | ~P[4,5,0]) & (~P[1,1,0] | ~P[5,1,0]) & (~P[1,1,0] | ~P[5,3,0]) & (~P[1,1,0] |
~P[5,5,0]) & (~P[1,1,0] | ~P[6,1,0]) & (~P[1,1,0] | ~P[6,3,0]) & (~P[1,1,0] | ~P[6,5,0]) & (~P[1,1,0] |
~P[7,1,0]) & (~P[1,1,0] | ~P[7,2,0]) & (~P[1,1,0] | ~P[7,3,0]) & (~P[1,1,0] | ~P[7,4,0]) & (~P[1,1,0] |
~P[7,5,0]) & (~P[1,2,0] | ~P[1,3,0]) & (~P[1,2,0] | ~P[1,5,0]) & (~P[1,2,0] | ~P[2,3,0]) & (~P[1,2,0] |
~P[2,5,0]) & (~P[1,2,0] | ~P[3,1,0]) & (~P[1,2,0] | ~P[3,3,0]) & (~P[1,2,0] | ~P[3,5,0]) & (~P[1,2,0] |
~P[4,1,0]) & (~P[1,2,0] | ~P[4,2,0]) & (~P[1,2,0] | ~P[4,3,0]) & (~P[1,2,0] | ~P[4,4,0]) & (~P[1,2,0] |
~P[4,5,0]) & (~P[1,2,0] | ~P[5,1,0]) & (~P[1,2,0] | ~P[5,3,0]) & (~P[1,2,0] | ~P[5,5,0]) & (~P[1,2,0] |
~P[6,1,0]) & (~P[1,2,0] | ~P[6,3,0]) & (~P[1,2,0] | ~P[6,5,0]) & (~P[1,2,0] | ~P[7,1,0]) & (~P[1,2,0] |
~P[7,2,0]) & (~P[1,2,0] | ~P[7,3,0]) & (~P[1,2,0] | ~P[7,4,0]) & (~P[1,2,0] | ~P[7,5,0]) & (~P[1,3,0] |
~P[1,5,0]) & (~P[1,3,0] | ~P[2,3,0]) & (~P[1,3,0] | ~P[2,5,0]) & (~P[1,3,0] | ~P[3,1,0]) & (~P[1,3,0] |
~P[3,3,0]) & (~P[1,3,0] | ~P[3,5,0]) & (~P[1,3,0] | ~P[4,1,0]) & (~P[1,3,0] | ~P[4,2,0]) & (~P[1,3,0] |
```

Pre-reading

From Propositional Logic to Classical Planning

Poll 2

For the blocks world predicate $on(block1, block2)$ how many grounded atoms (facts), do we have with 4 block objects?



A B C D

on(A, A) ✓

on(A, B) ✓

A, C

~~A, D~~

B, A

¬on(A, B)

First-Order Logic and Grounding

We're not going to get into first order logic, but it has some ideas that we'll use in planning

Objects (things not Boolean symbols)

A B C

Predicate functions (input objects, evaluate to Boolean)

inHand(A)

Variables and $\forall x$

inHand(block)

inHand(C)

Solving first-order logic problems

- Not magic ☹ Still need to ground predicates and objects into propositional logic symbols

Search, Logic, and Classical Planning

State-space Search Planning

env

- Assumes actions and transitions are provided for you, s' = result(s, a)
- State changes as you take actions

Propositional Logic Planning

- Can reason about what actions are possible and their effects
- Represent world with only Boolean symbols
- Different symbols for different time points

Classical Planning

- Can reason about what actions are possible and their effects
- State changes as you take actions

Autonomous Planning
- MDP
- RL
- State space search
- Classical Planning

Pre-reading

Classical Planning (or Symbolic Planning)

Representation

Idea of Classical Planning

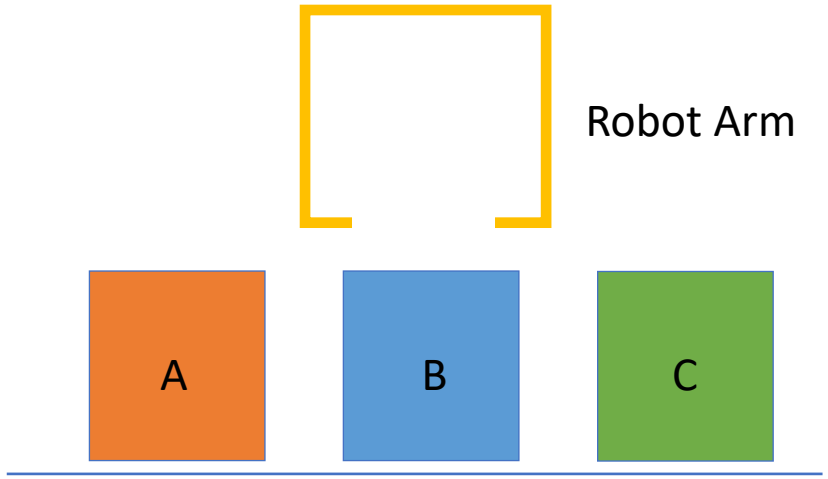
Represent objects/values separately from the state (instances)

Define predicates as true/false functions over the objects (propositions)

States are conjunctions of predicates

Goals are conjunctions of predicates

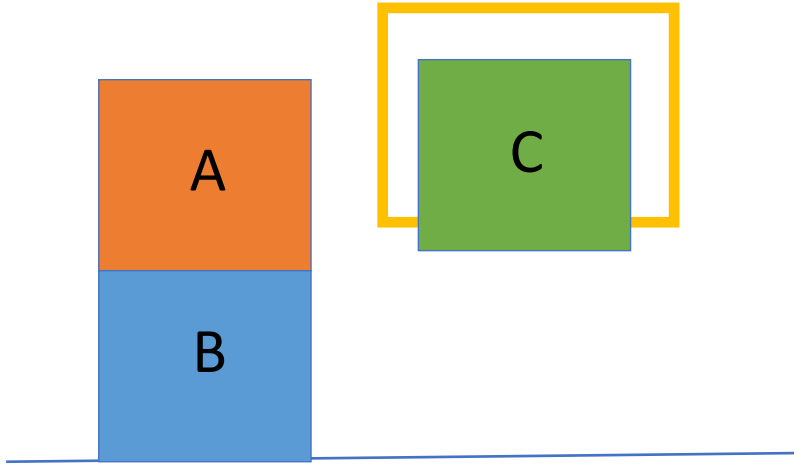
Operators (actions)



Poll 3

Classical planning for blocks world. For the start state below, which facts are part of the state representation? Select all that apply

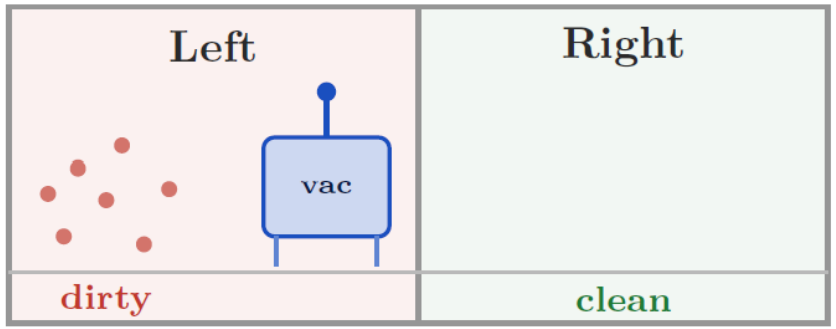
1) In-Hand(C)
2) ¬On-Table(A)
3) On-Table(B)
4) ¬On-Table(C)
5) ¬On-Block(B,C)
6) On-Block(A,B)
7) ¬HandEmpty()



Take 1: ~50% selected all → Incorrect

Factored State Representation

From pre-reading



Count the situations first. Two choices for where the robot is, two for whether Left is dirty, two for whether Right is dirty, so $2 \times 2 \times 2 = 8$ altogether. Now describe them both ways.

Atomically. Number the eight situations $s_1, \ldots, s_8$ and record the transitions in a table: from s_1, *Suck* leads to s_5; from s_1, *MoveRight* leads to s_2; and so on for all 8×3 entries. Nothing is wrong with this description, but look at what it costs and what it hides. Somebody has to write the table out by hand, entry by entry. Nothing in it explains why *Suck* from s_1 lands on s_5 and not somewhere else. And adding a second robot gives $2 \times 2 \times 2 \times 2 = 16$ situations and an entirely new table, since none of the old numbers mean what they used to.

Factored. Name three variables:

$$RobotIn \in \{Left, Right\}, \qquad DirtLeft \in \{\mathsf{true}, \mathsf{false}\}, \qquad DirtRight \in \{\mathsf{true}, \mathsf{false}\}.$$

Closed World Assumption (CWA)

Closed world assumption (CWA):

Any fact not listed in a state is false in that state

(Factored) State representation:

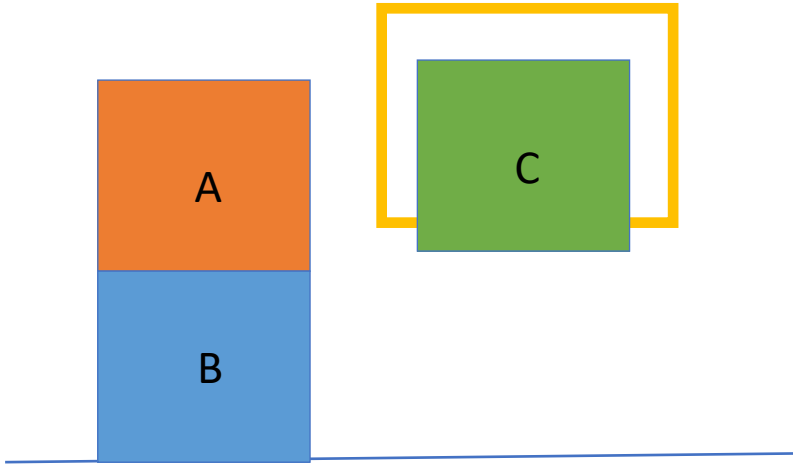
In-Hand(C)

On-Table(B)

On-Block(A,B)

Clear(A)

Clear(C)



Not explicitly included:

~HandEmpty(), ~On-Table(C), ~On-Table(A), ~On-Block(B,A), ~On-Block(C,A), ~On-Block(B,C), ~On-Block(C,B), ~On-Block(A,C), ~Clear(B), ~In-Hand(A), ~In-Hand(B)

Poll 4

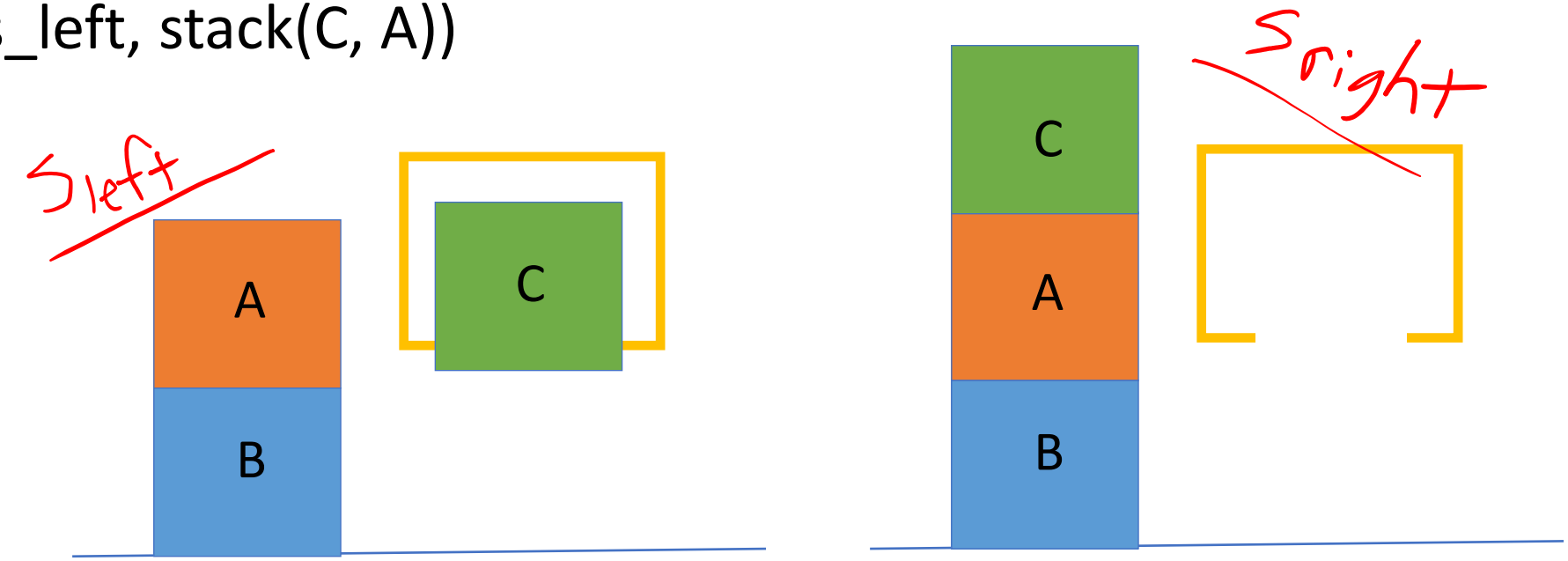
Classical planning for blocks world. When we take an action to move from the left state to the right state what do we do?

A. Change current state from s_{left} to s_{right}

B. Add On(C, A), HandEmpty()

C. Delete Clear(A), InHand(C) and Add On(C, A), HandEmpty()

D. Call env.result(s_left, stack(C, A))



STRIPS Representation

STRIPS = Stanford Research Institute Problem Solver (1970s system)

Actions defined by:

- Preconditions (only positive facts)
- Add effects
- Delete effects

Convention: all other facts remain unchanged → avoids frame axioms

Actions

Actions change the state by adding/deleting predicates

Preconditions:

Actions can be applied only if all precondition predicates are true in the current state

Effects:

New state is a copy of the current predicates with the addition or deletion of specified predicates

Unlike the successor-state axioms, we do not explicitly represent time

The Frame Problem

Frame problem: how to specify what remains unchanged after actions

- Frame axioms (logic): one per unaffected predicate
- Successor-State Axiom: combines persistence + change
- STRIPS convention: actions only list what changes (preconditions, add, delete)

Pre-reading

Blocks World Predicates and Operators

Rules of Blocks World

Blocks are picked up and put down by the hand

Blocks can be picked up only if they are clear

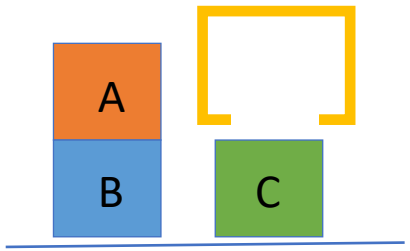
Hand can pick up a block only if the hand is empty

Hand can pick up and put down blocks on blocks or on the table

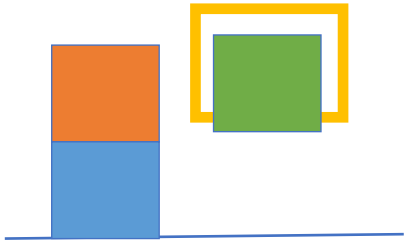
Pickup Block C from Table (State Transition)

Instances:
Blocks A, B, C

Possible Predicates:
HandEmpty()
On-Table(block)
On-Block(b1,b2)
Clear(block)
In-Hand(block)



<u>State:</u>
HandEmpty()
On-Table(B)
On-Table(C)
On-Block(A,B)
Clear(A)
Clear(C)

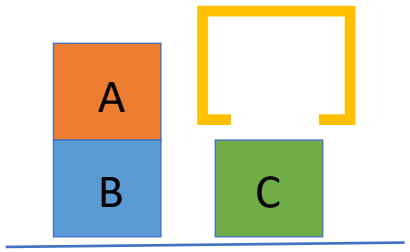


<u>State:</u>
In-Hand(C)
On-Table(B)
On-Block(A,B)
Clear(A)
Clear(C)

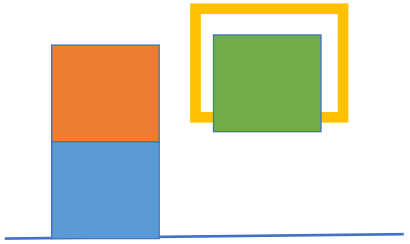
Pickup Block C from Table (Preconditions, Effects)

Instances:
Blocks A, B, C

Possible Predicates:
HandEmpty()
On-Table(block)
On-Block(b1,b2)
Clear(block)
In-Hand(block)



<u>State:</u>
HandEmpty()
On-Table(B)
On-Table(C)
On-Block(A,B)
Clear(A)
Clear(C)



<u>State:</u>
In-Hand(C)
On-Table(B)
On-Block(A,B)
Clear(A)
Clear(C)
Delete HandEmpty()
Delete On-Table(C)

Operator: Pickup-Block-C from Table

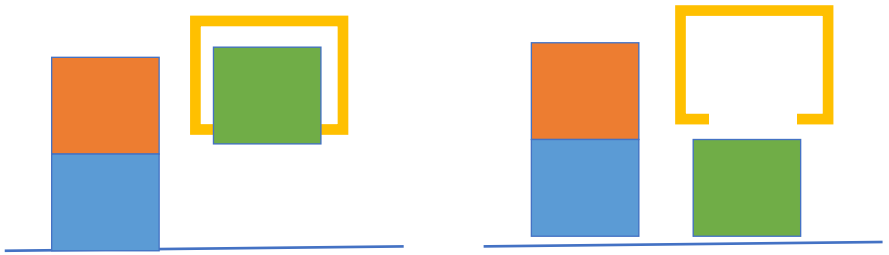
Preconditions	Effects
HandEmpty()	Add In-Hand(C)
Clear(C)	Delete HandEmpty()
On-Table(C)	On-Table(C)

Operator: Pickup-Block from Table

Preconditions	Effects
HandEmpty()	Add In-Hand(block)
Clear(block)	Delete HandEmpty()
On-Table(block)	On-Table(block)

Create a variable that takes on the value of a particular instance for all times it appears in an operator.

Operator: PutDown-Block on Table



Preconditions	Effects	
In-Hand(block)	Add	HandEmpty()
		On-Table(block)
	Delete	In-Hand(block)

Why don’t we need to check if ~HandEmpty() is true?

Operators for Block Stacking

Pickup_Table(b):

Pre: HandEmpty(), Clear(b), On-Table(b)

Add: In-Hand(b)

Delete: HandEmpty(), On-Table(b)

Pickup_Block(b,c):

Pre: HandEmpty(), On-Block(b,c), b!=c

Add: In-Hand(b), Clear(c)

Delete: HandEmpty(), On-Block(b,c)

Putdown_Table(b):

Pre: In-Hand(b)

Add: HandEmpty(), On-Table(b)

Delete: In-Hand(b)

Putdown_Block(b,c):

Pre: In-Hand(b), Clear(c)

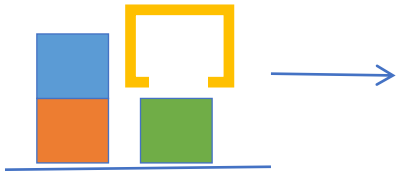
Add: HandEmpty(), On-Block(b,c)

Delete: Clear(c), In-Hand(b)

Why do we need separate operators for table vs on a block?

Example Matching Operators

HandEmpty() & On-Table(O) & On-Block(B,O) & Clear(B) & On-Table(G) & Clear(G)

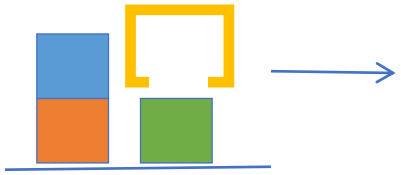


Example Matching Operators

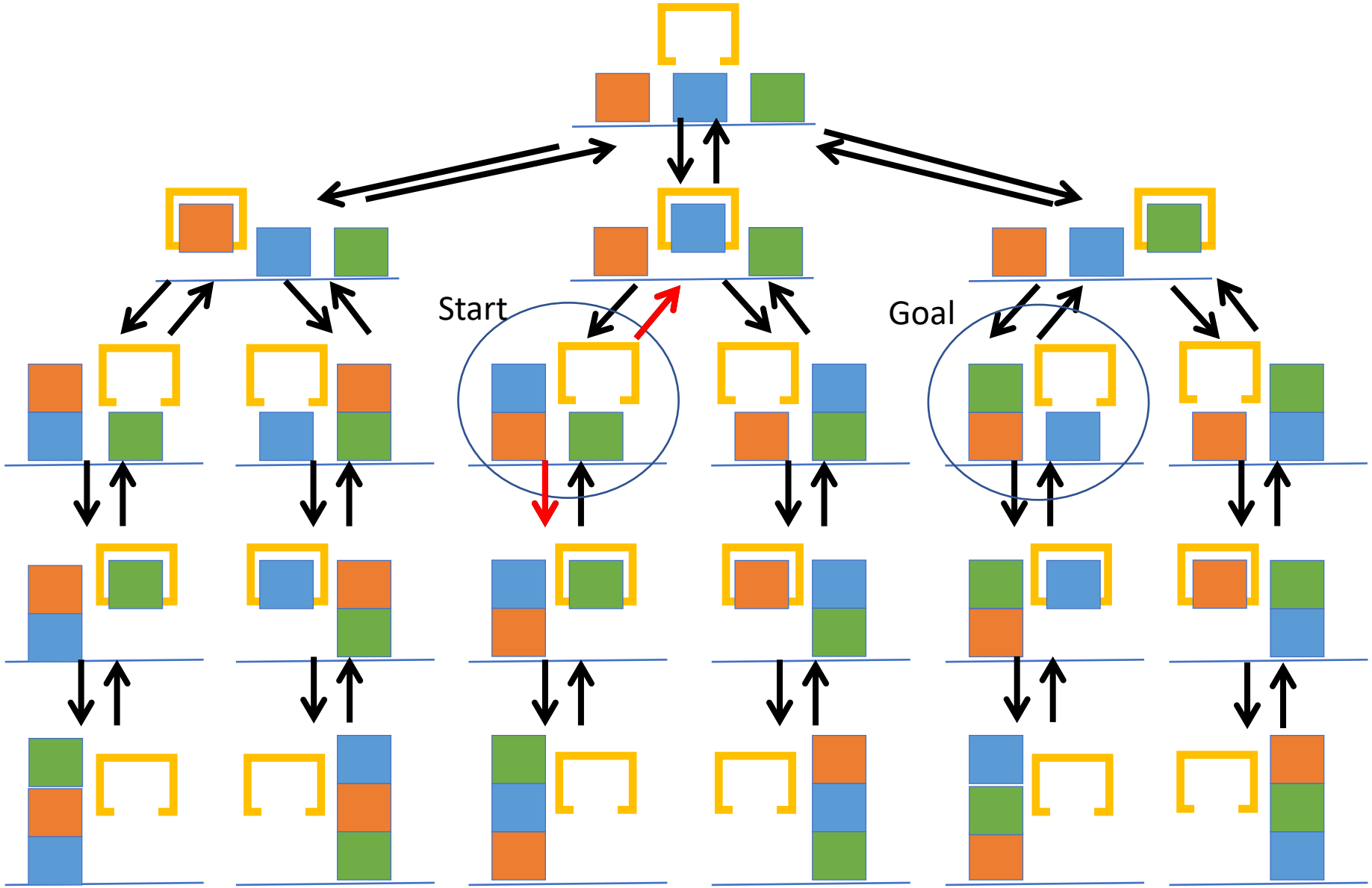
HandEmpty() & On-Table(O) & On-Block(B,O) & Clear(B) & On-Table(G) & Clear(G)

Pickup_Block(b,c):
Pre: HandEmpty(), On-Block(b,c), b!=c
Add: In-Hand(b), Clear(c)
Delete: HandEmpty(), On(b,c)

Pickup_Table(b):
Pre: HandEmpty, Clear(b), On-Table(b)
Add: In-Hand(b)
Delete: HandEmpty(), On-Table(b)



State Space Graph (also called Reachability Graph)



Example Matching Operators

HandEmpty() & On-Table(O) & On-Block(B,O) & Clear(B) & On-Table(G) & Clear(G)

Pickup_Block(B,O)

On-Table(O) & Clear(B) & On-Table(G) & Clear(G) & In-Hand(B) & Clear(O)

Putdown_Table(B)

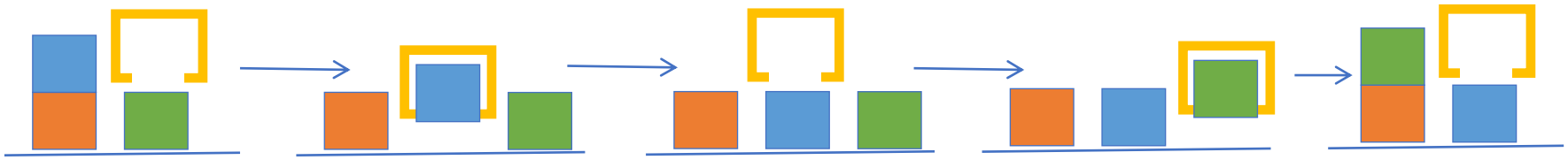
On-Table(O) & Clear(O) & On-Table(G) & Clear(G) & Clear(B) & On-Table(B) & HandEmpty()

Pickup_Table(G)

On-Table(O) & Clear(B) & Clear(G) & Clear(O) & On-Table(B) & In-Hand(G)

Putdown_Block(G,O)

On-Table(O) & Clear(B) & Clear(G) & On-Table(B) & On-Block(G,O) & HandEmpty()



Pre-reading

Classical Planning Representation

PDDL

Poll 5

Which of the following go into the PDDL problem file (rather than the domain file)?

A. Objects
B. Start state
C. Predicates
D. Actions
E. Goal
F. None of the above

PDDL

Domain file

```
(define (domain blocksworld)
  (:requirements :strips)
  (:predicates (on ?x ?y) (onTable ?x) (clear ?x) (holding ?x) (handEmpty))
  (:action stack
    :parameters (?x ?y)
    :precondition (and (holding ?x) (clear ?y))
    :effect (and (on ?x ?y) (clear ?x) (handEmpty)
            (not (holding ?x)) (not (clear ?y)))))
```

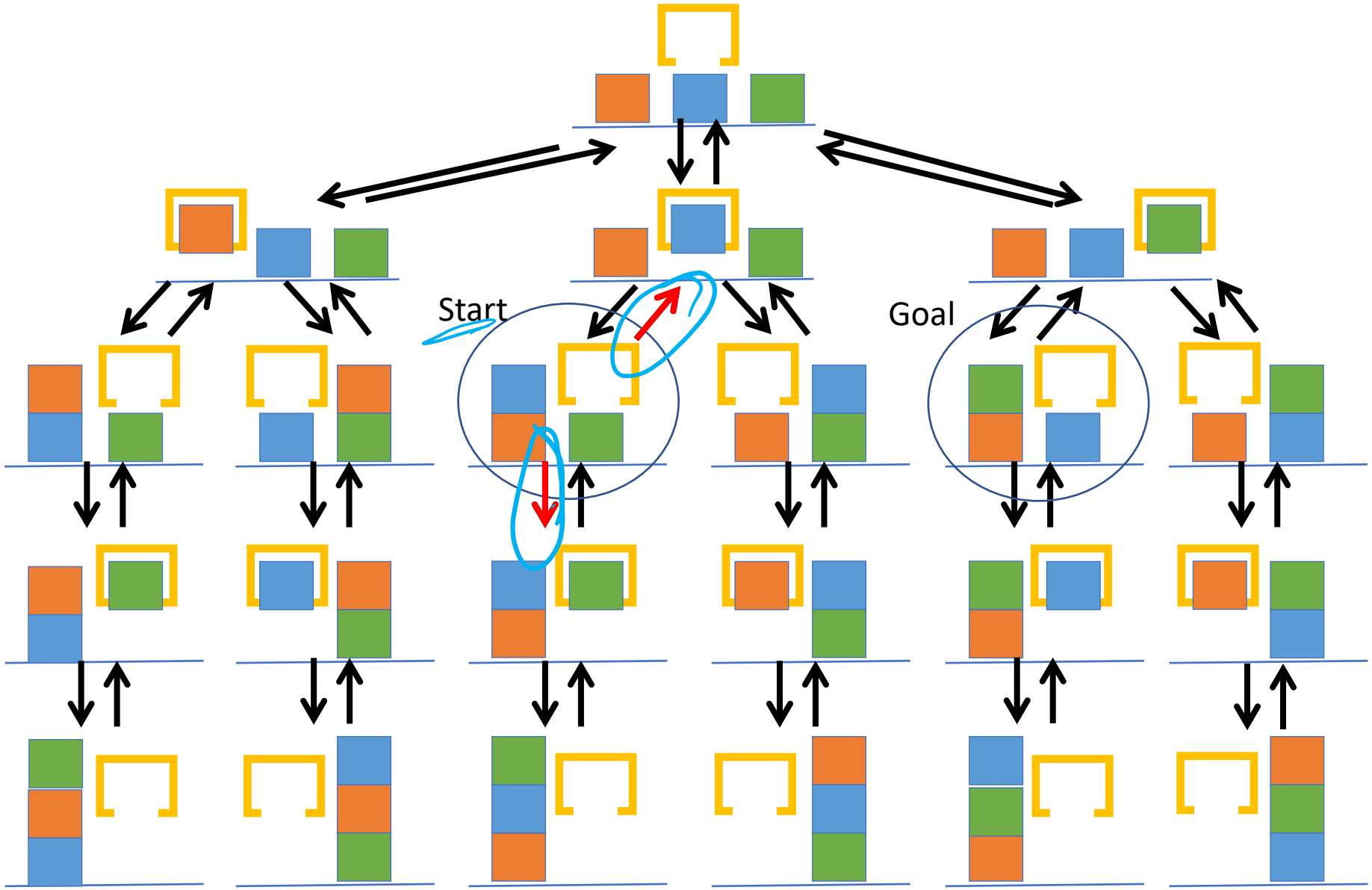
Problem file

```
(define (problem bw-3blocks)
  (:domain blocksworld)
  (:objects a b c)
  (:init (on c a) (onTable a) (onTable b) (clear c) (clear b) (handEmpty))
  (:goal (and (on a b) (on b c))))
```

Solivng Planning Problems

State-space Search

Search with a State Space Graph



Finding Plans with Symbolic Representations

Breadth-First Search

Sound? Yes

Complete? Yes

Optimal? Yes

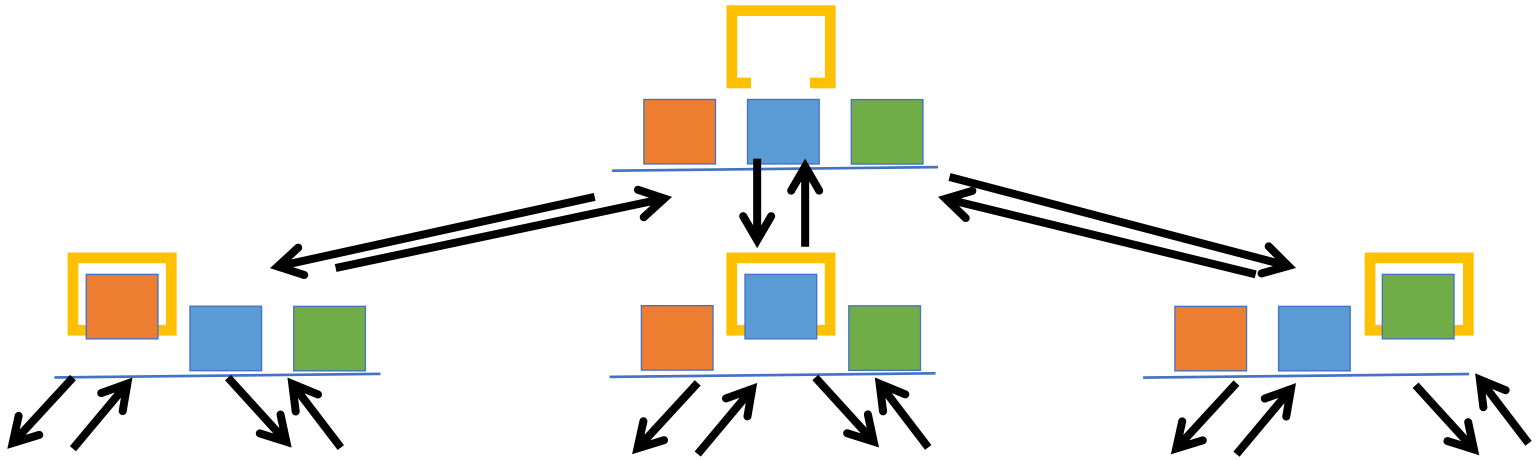
Soundness - all solutions found are legal plans

Completeness - a solution can be found whenever one actually exists

Optimality - the order in which solutions are found is consistent with some measure of plan quality

Size of the State-Space Tree

A state-space tree's size is exponential in the number of predicates



Can we reduce the size of the graph we use for planning?

Today

Polls covering Pre-reading

- Planning with Propositional Logic
- From Propositional Logic to Classical Planning
- Classical Planning Representation
- Solving Planning Problems: State space search

New

- Solving Planning Problems: Planning Graphs and GraphPlan

Next time

- From GraphPlan to Modern Planning
- Planning with Continuous States

Solivng Planning Problems

Planning Graphs & Graph Plan

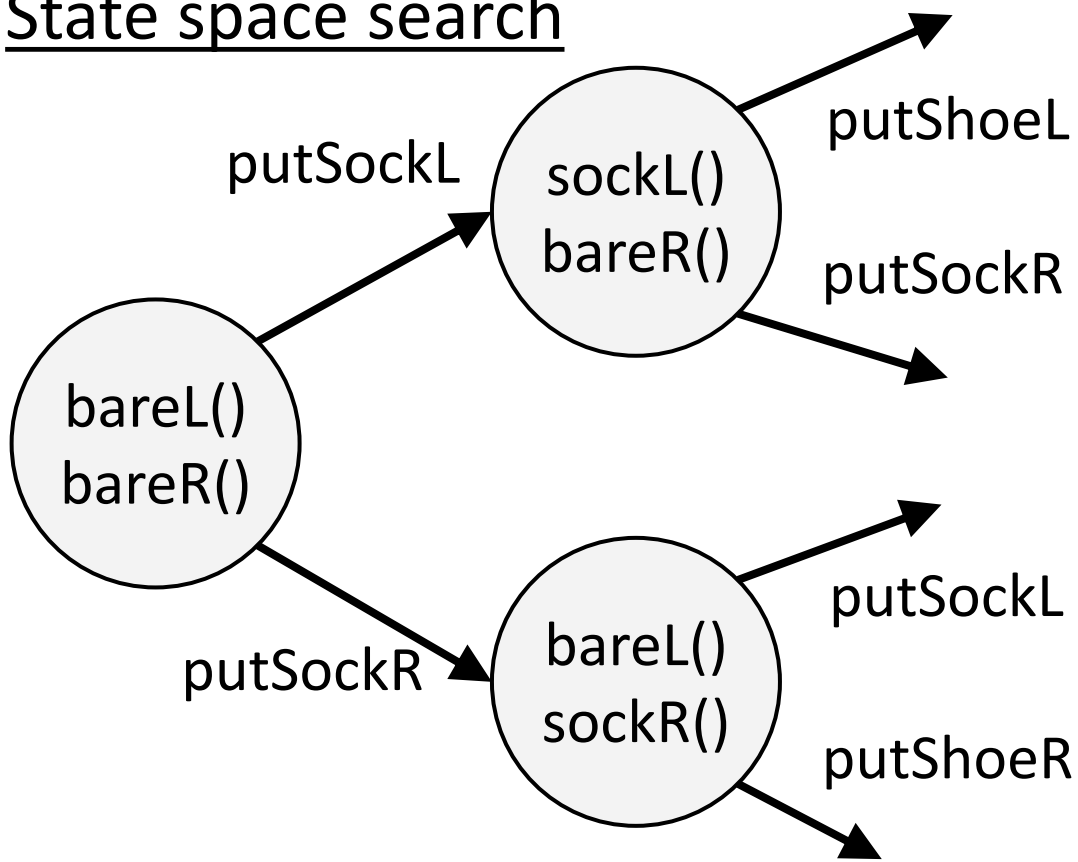
GraphPlan

1) Allow actions to be simultaneous
2) Always add predicates (don't delete)

GraphPlan is a relaxation of other classical planning search techniques

The GraphPlan search graph space is linear in the number of predicates

<u>State space search</u>

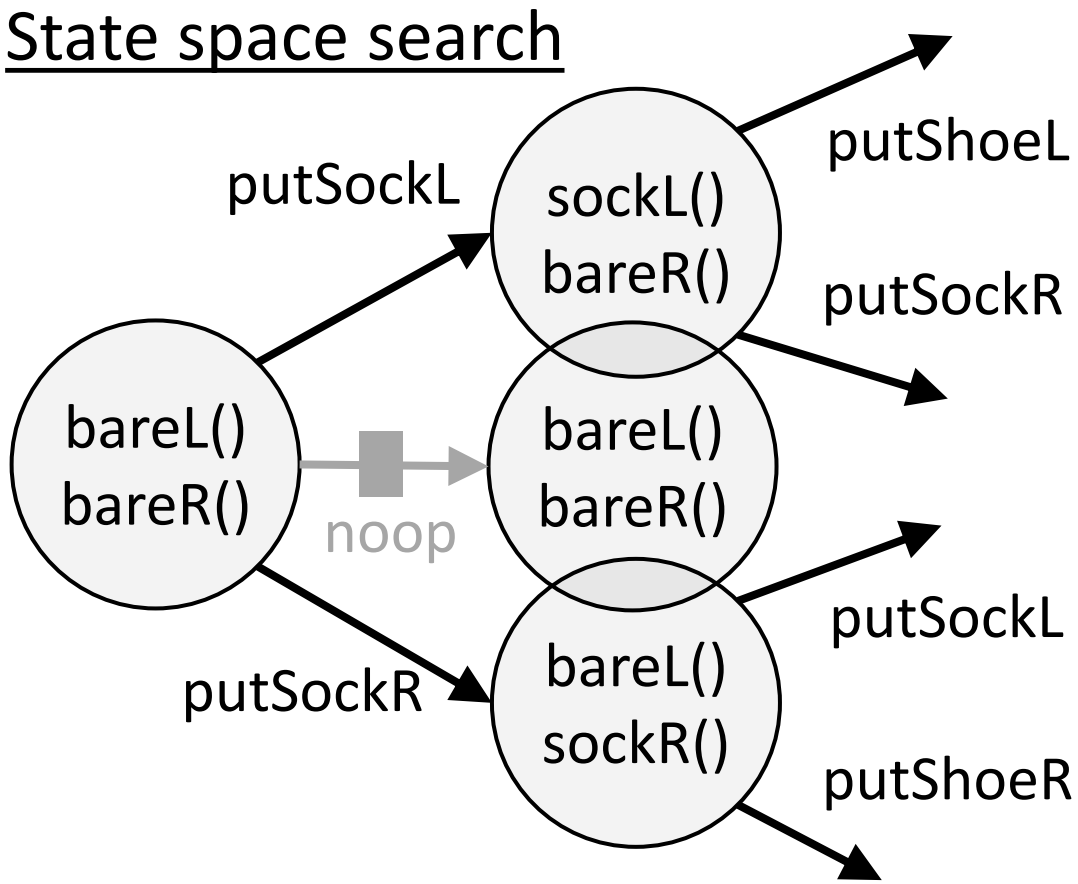


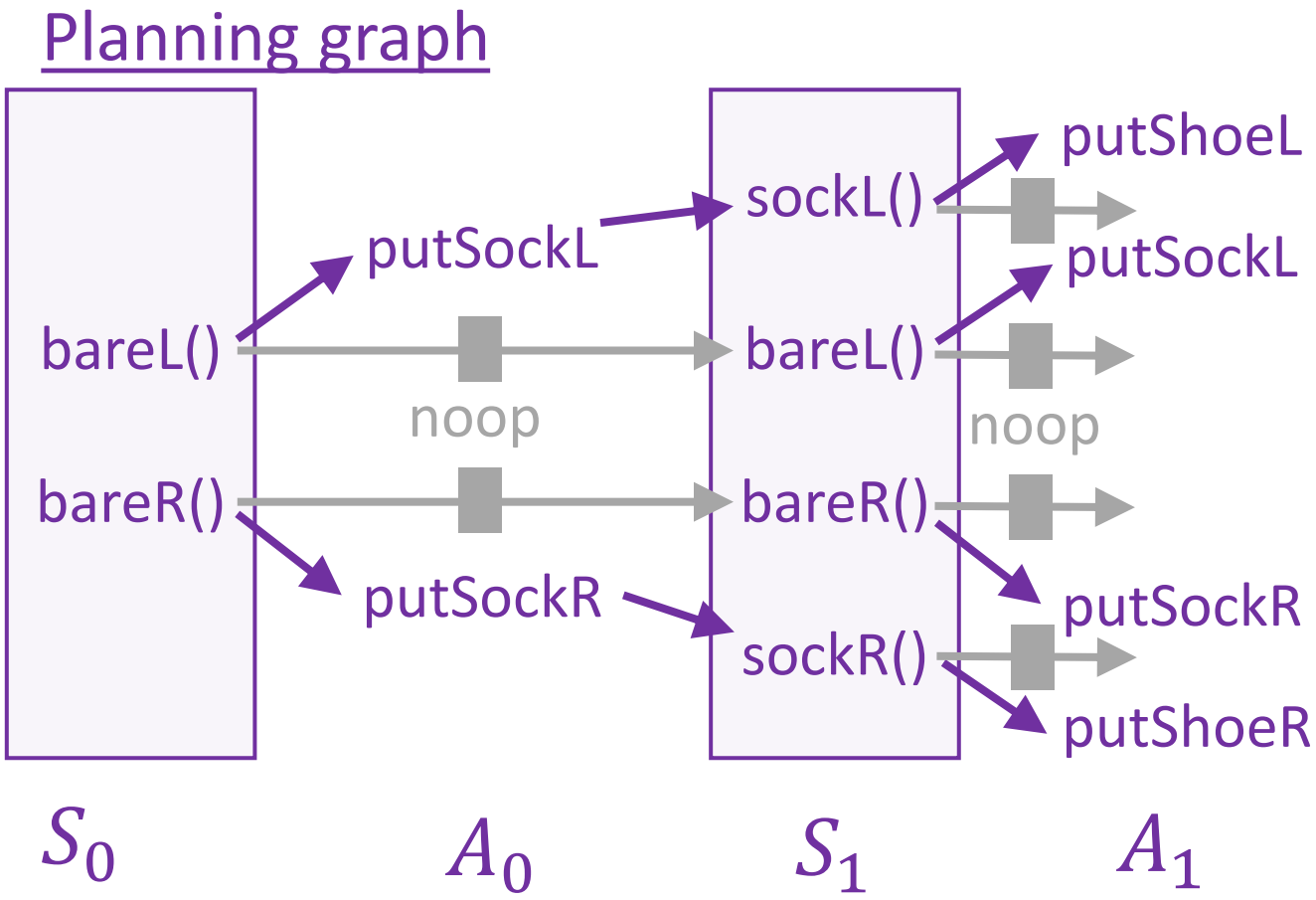
GraphPlan

1) Allow actions to be simultaneous
2) Always add predicates (don't delete)

GraphPlan is a relaxation of other classical planning search techniques

The planning graph is linear in the number of predicates





Building a Planning Graph

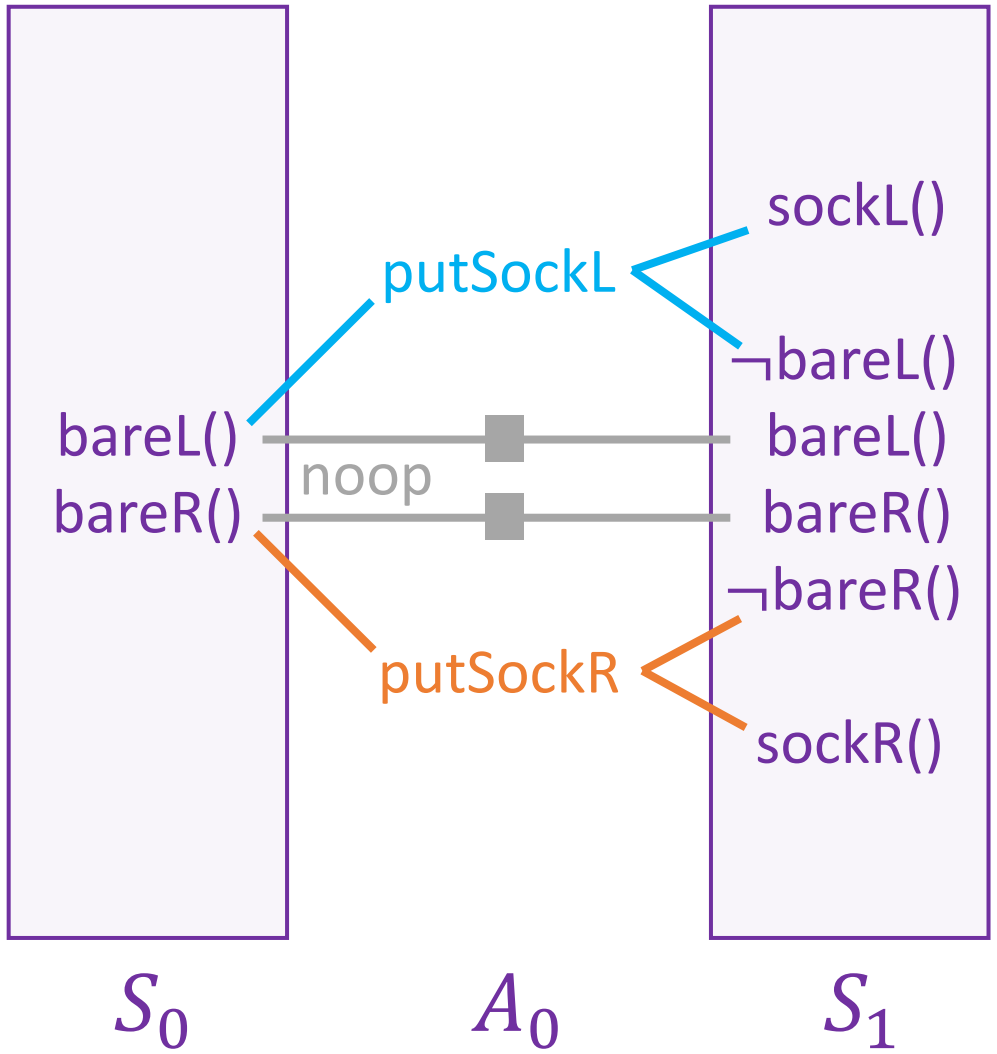
Initialize S_0 with all predicates in the start state

bareL()
bareR()

S_0 A_0 S_1

Building a Planning Graph

Extend graph to S_1 with all actions in A_0 that can be taked from S_0



For now, we are just assuming that any actions (including no-ops) can be taken individually in any order to get us to the next state

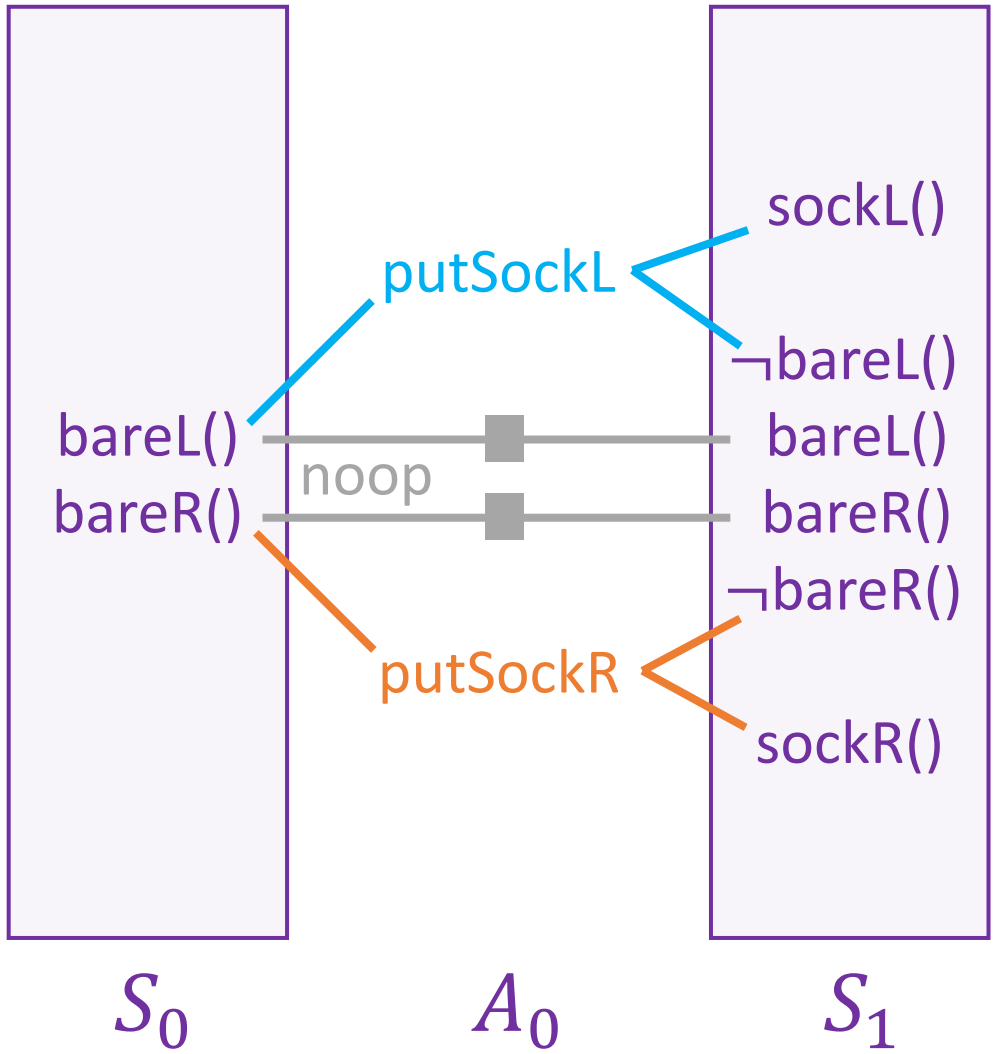
→ This certainly isn't always true

E.g., we can't put our socks on and then take the no-ops that require bare feet

(More on these exclusion checks later)

Building a Planning Graph

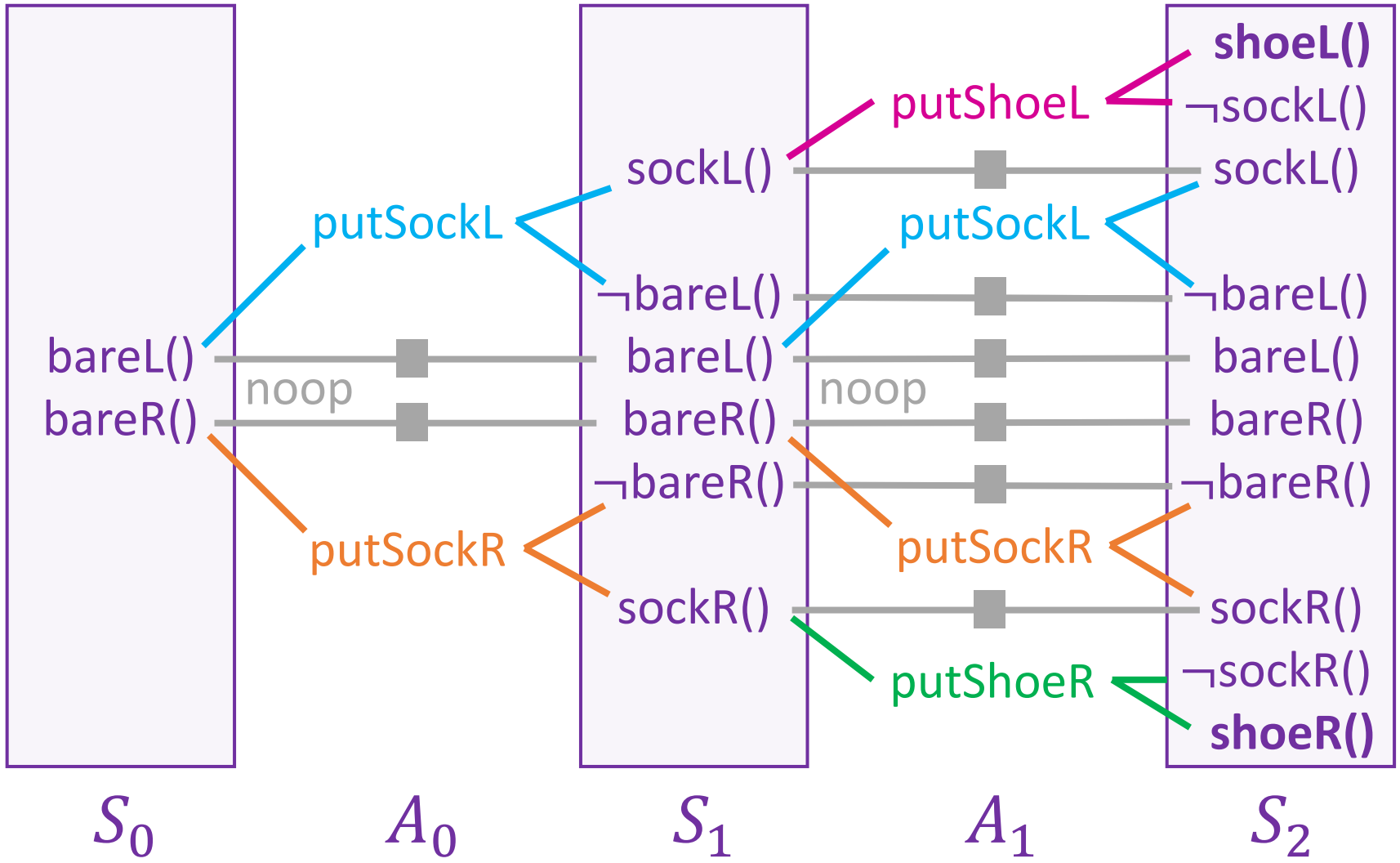
Search for solution. Does S_1 contain all goal propositions, shoeL(), shoeR()?



Nope, not yet

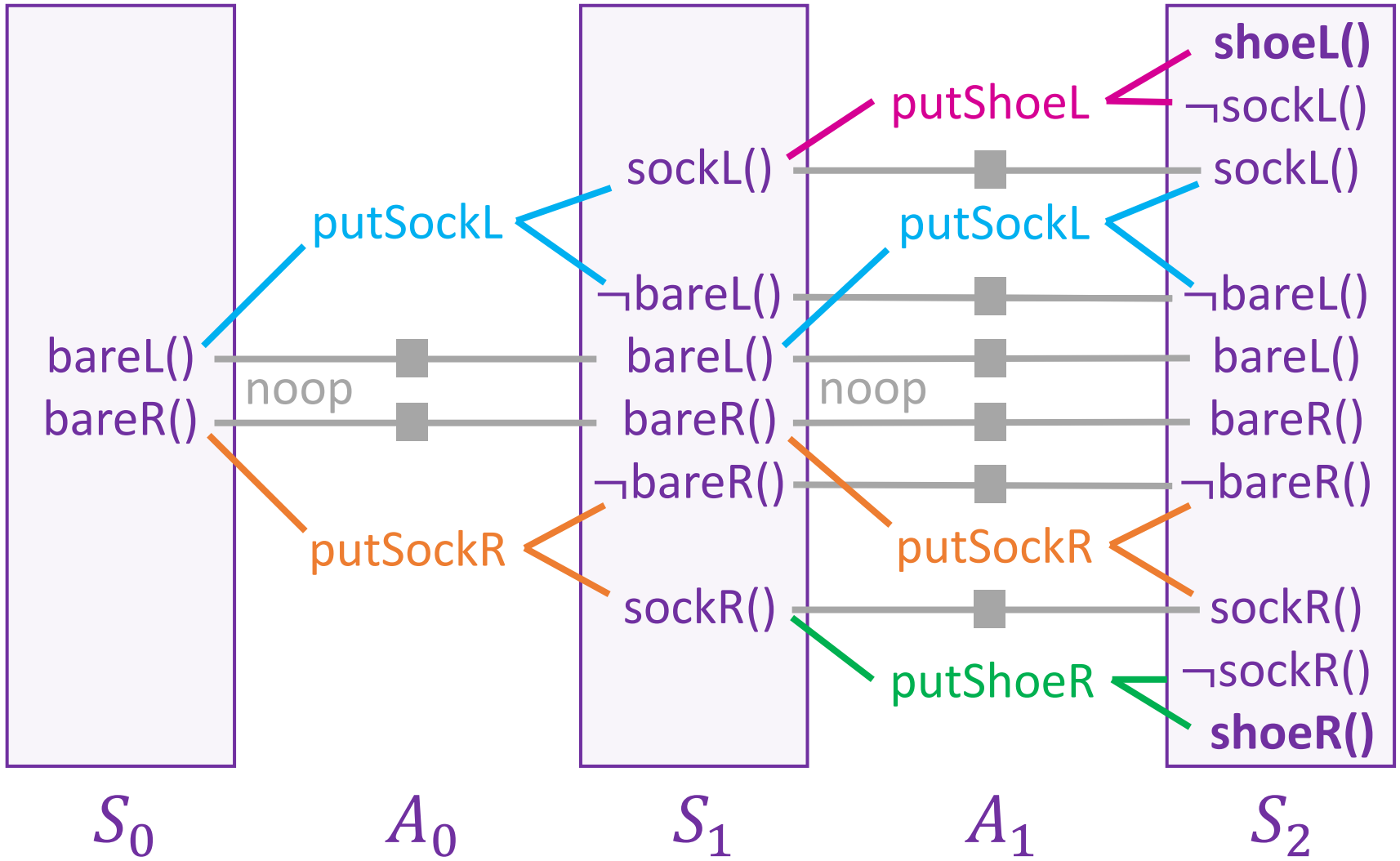
Building a Planning Graph

Extend graph to S_2 with all actions in A_1 that can be taked from S_1



Building a Planning Graph

Search for solution. Does S_2 contain all goal propositions, **shoeL()**, **shoeR()**?



Yes, but
....is this ok??

Maybe ☺

We need to check that we can actually take the subset of actions that lead us the goal proposition