Lecture 22

CS 6260 Automated Planning and Learning

Learning Object-Centred Models III

Department of Computer Science and Engineering

Indian Institute of Technology Madras

Recap: LOCM

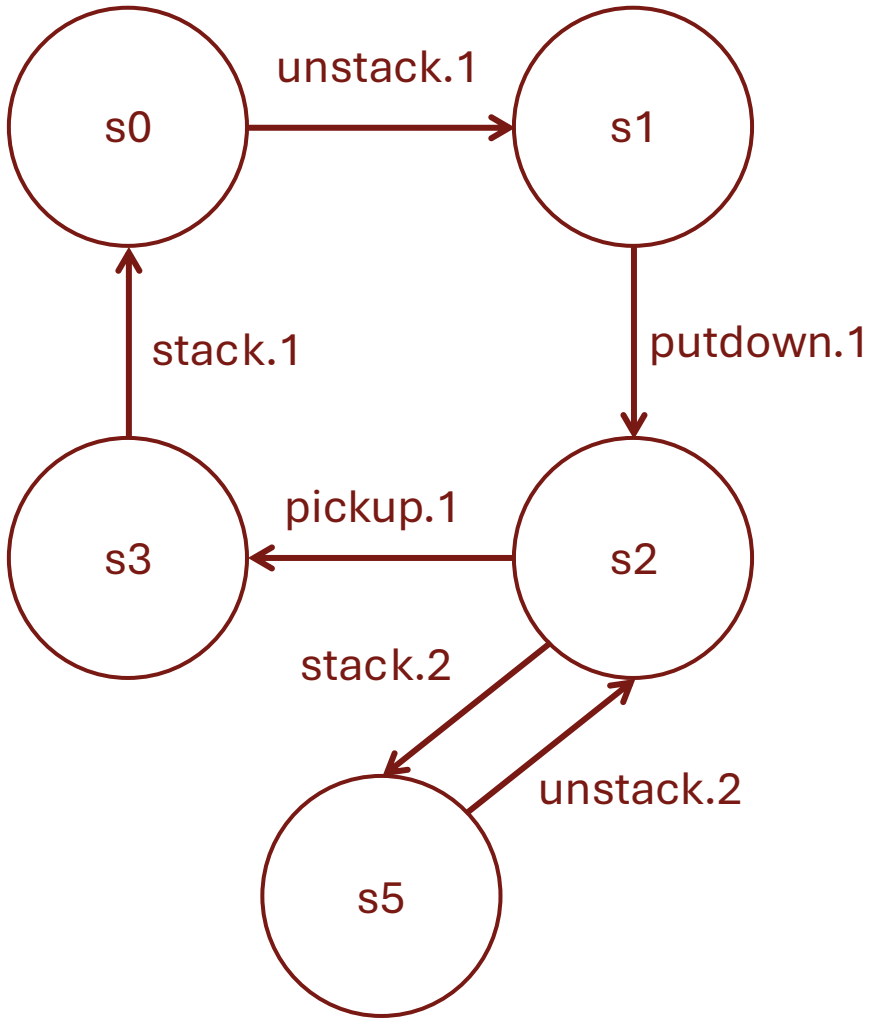
Inducing FSMs

0	Sort Formation	
1	Induce FSMs	
2	Zero Analysis	
3	Parameterize FSMs	
4	Create & Merge Params	
5	Remove Flawed Params	
6	Static Preconditions	
7	Output PDDL Schema	

	Transition	Start	End	Reason
A	unstack.1	S_0	S_1	first time seen, assign $S_0 \rightarrow S_1$
	putdown.1	S_1	S_2	Continuity: start must = end of prev (S_1). Assign fresh end S_2
	pickup.1	S_2	S_3	Continuity: start = S_2. Assign fresh end S_3
	stack.1	S_3	~~S_4~~=S_0	Continuity: start = S_3. Assign fresh end S_4
B	unstack.2	S_5	~~S_6~~=S_2	first time seen, assign $S_5 \rightarrow S_6$
	pickup.1	~~S_6~~ S_2	~~S_7~~ S_3	Using 1-1 rule
	stack.1	S_3	~~S_4~~=S_0	
	unstack.1	S_0	S_1	
	putdown.1	S_1	S_2	
	stack.2	S_2	~~S_7~~=S_5	
C	stack.2	S_2	~~S_7~~=S_5	
	unstack.2	S_5	S_2	

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B

Inducing FSMs

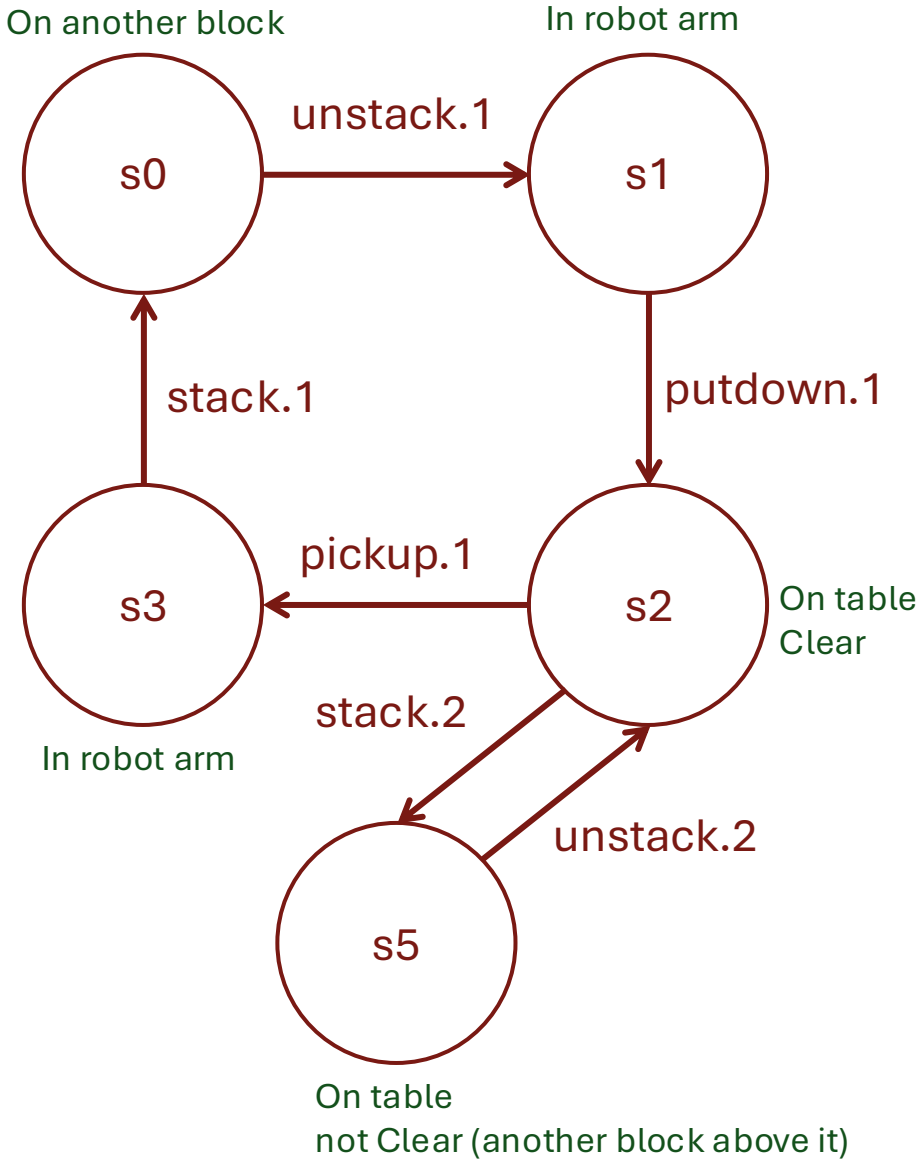


	Transition	Start	End
A	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	pickup.1	S_2	S_3
	stack.1	S_3	~~S_4~~=S_0
B	unstack.2	S_5	~~S_6~~=S_2
	pickup.1	~~S_6~~ S_2	~~S_7~~ S_3
	stack.1	S_3	~~S_4~~=S_0
	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	stack.2	S_2	~~S_7~~=S_5
C	stack.2	S_2	~~S_7~~=S_5
	unstack.2	S_5	S_2

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B

Inducing FSMs

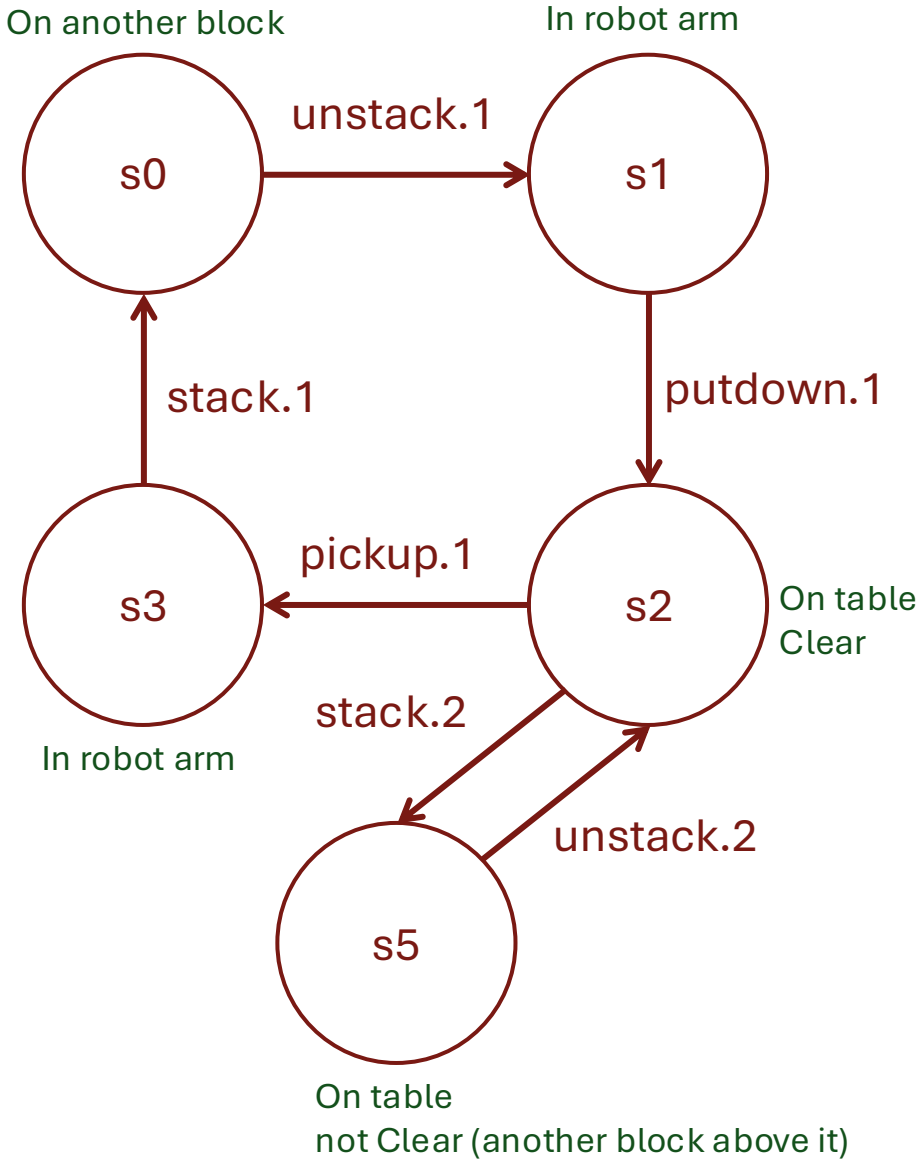


	Transition	Start	End
A	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	pickup.1	S_2	S_3
	stack.1	S_3	~~S_4~~=S_0
B	unstack.2	S_5	~~S_6~~=S_2
	pickup.1	~~S_6~~ S_2	~~S_7~~ S_3
	stack.1	S_3	~~S_4~~=S_0
	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	stack.2	S_2	~~S_7~~=S_5
C	stack.2	S_2	~~S_7~~=S_5
	unstack.2	S_5	S_2

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B

Inducing FSMs

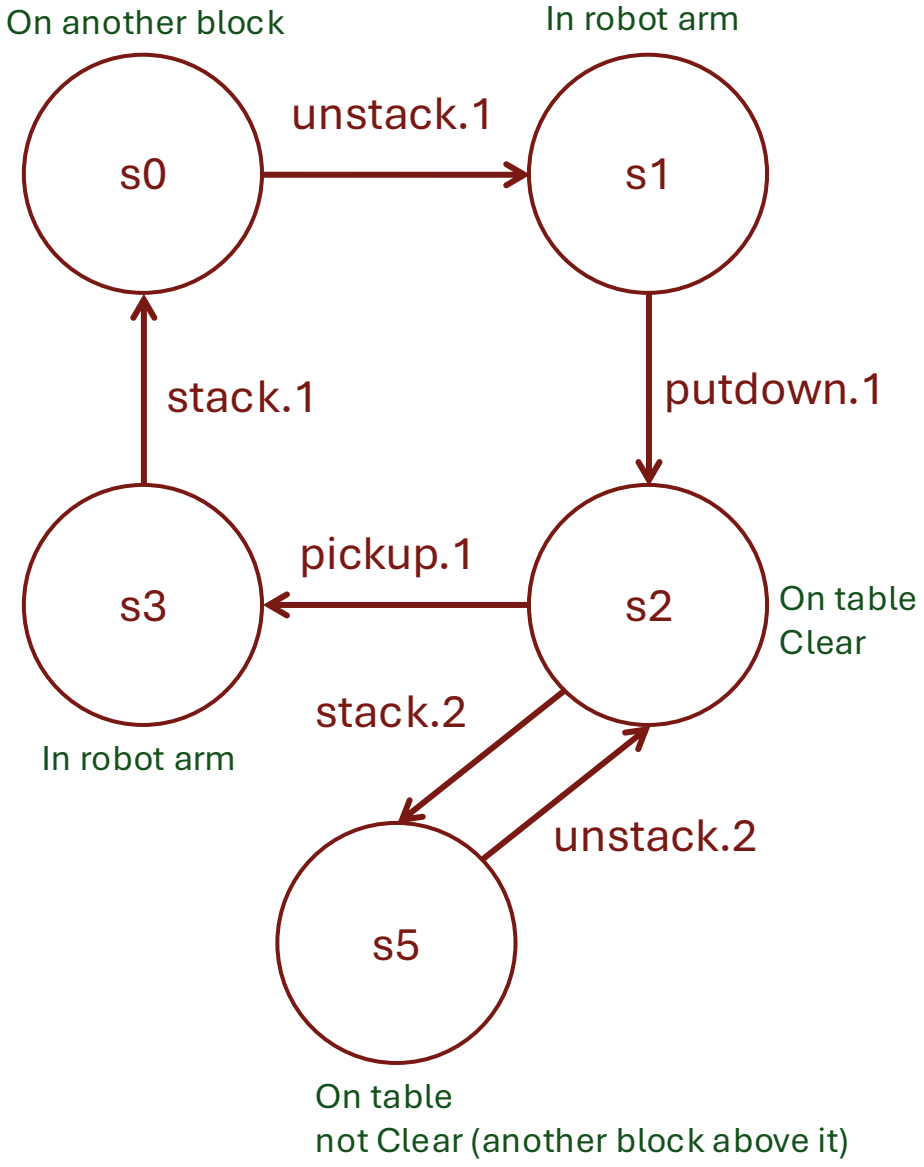


	Transition	Start	End
A	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	pickup.1	S_2	S_3
	stack.1	S_3	~~S_4~~=S_0
B	unstack.2	S_5	~~S_6~~=S_2
	pickup.1	~~S_6~~ S_2	~~S_7~~ S_3
	stack.1	S_3	~~S_4~~=S_0
	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	stack.2	S_2	~~S_7~~=S_5
C	stack.2	S_2	~~S_7~~=S_5
	unstack.2	S_5	S_2

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	pickup	C	-
10	putdown	C	-

Inducing FSMs

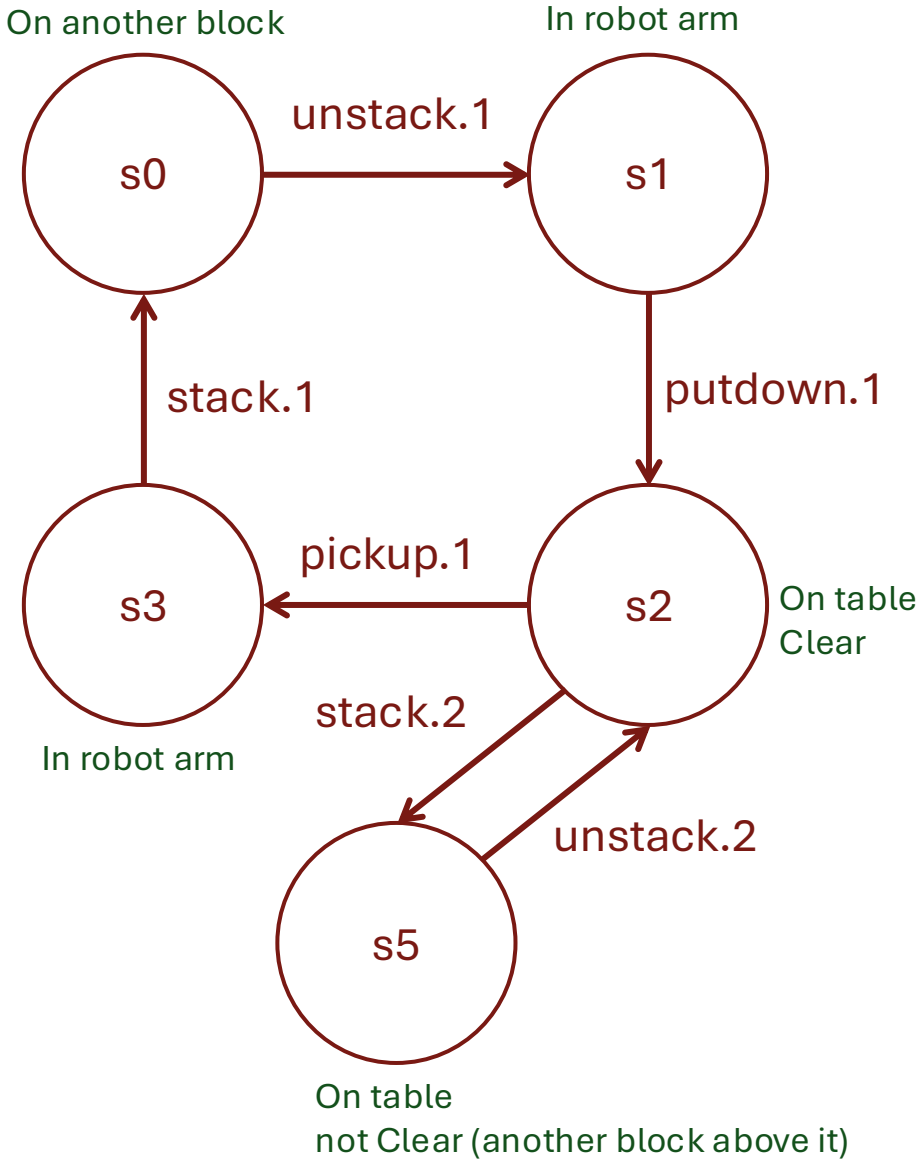


	Transition	Start	End
A	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	pickup.1	S_2	S_3
	stack.1	S_3	~~S_4~~=S_0
B	unstack.2	S_5	~~S_6~~=S_2
	pickup.1	~~S_6~~ S_2	~~S_7~~ S_3
	stack.1	S_3	~~S_4~~=S_0
	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	stack.2	S_2	~~S_7~~=S_5
C	stack.2	S_2	~~S_7~~=S_5
	unstack.2	S_5	S_2
	pickup.1	S_2	S_3

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	pickup	C	-
10	putdown	C	-

Inducing FSMs

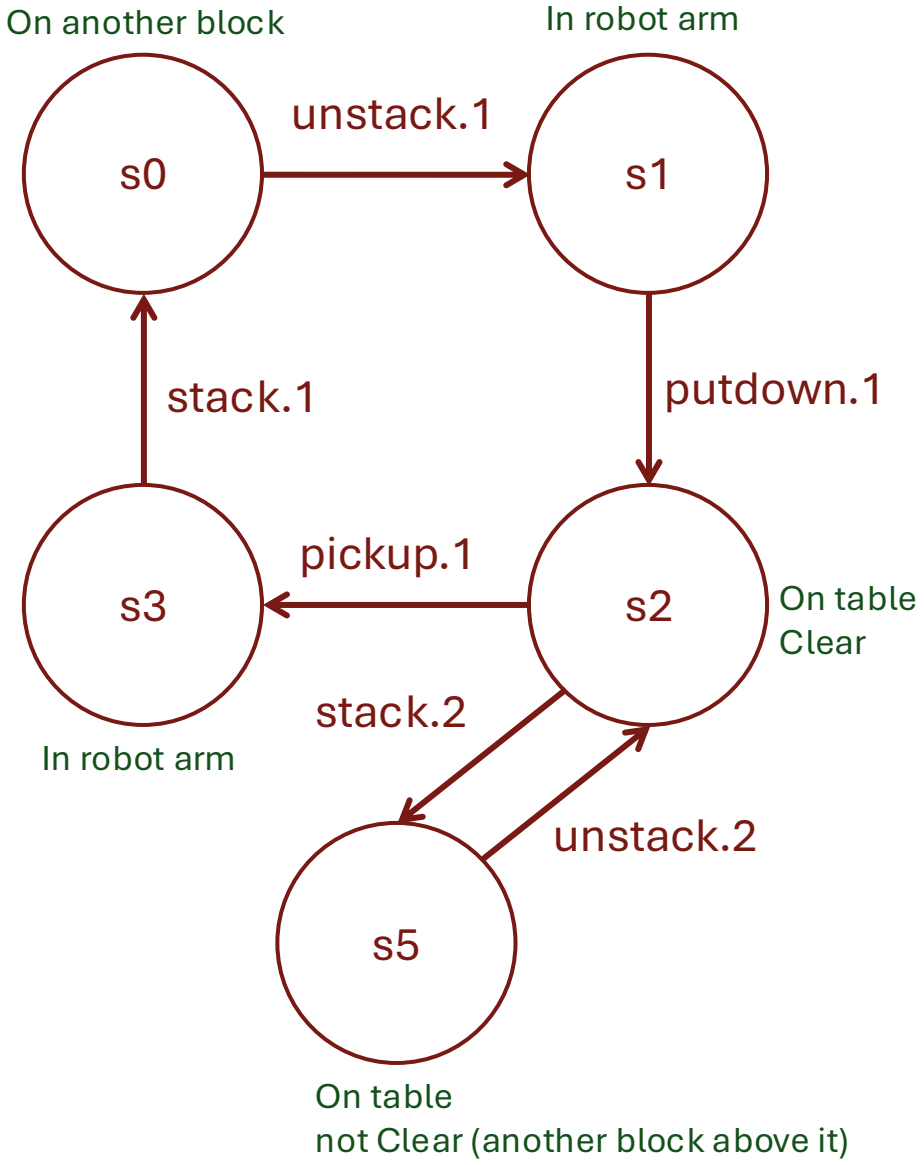


	Transition	Start	End
A	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	pickup.1	S_2	S_3
	stack.1	S_3	✖ S_4=S_0
B	unstack.2	S_5	✖ S_6=S_2
	pickup.1	✖ S_6 S_2	✖ S_7 S_3
	stack.1	S_3	✖ S_4=S_0
	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	stack.2	S_2	✖ S_7=S_5
C	stack.2	S_2	✖ S_7=S_5
	unstack.2	S_5	S_2
	pickup.1	S_2	S_3
	putdown.1	S_1	S_2

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	pickup	C	-
10	putdown	C	-

Inducing FSMs

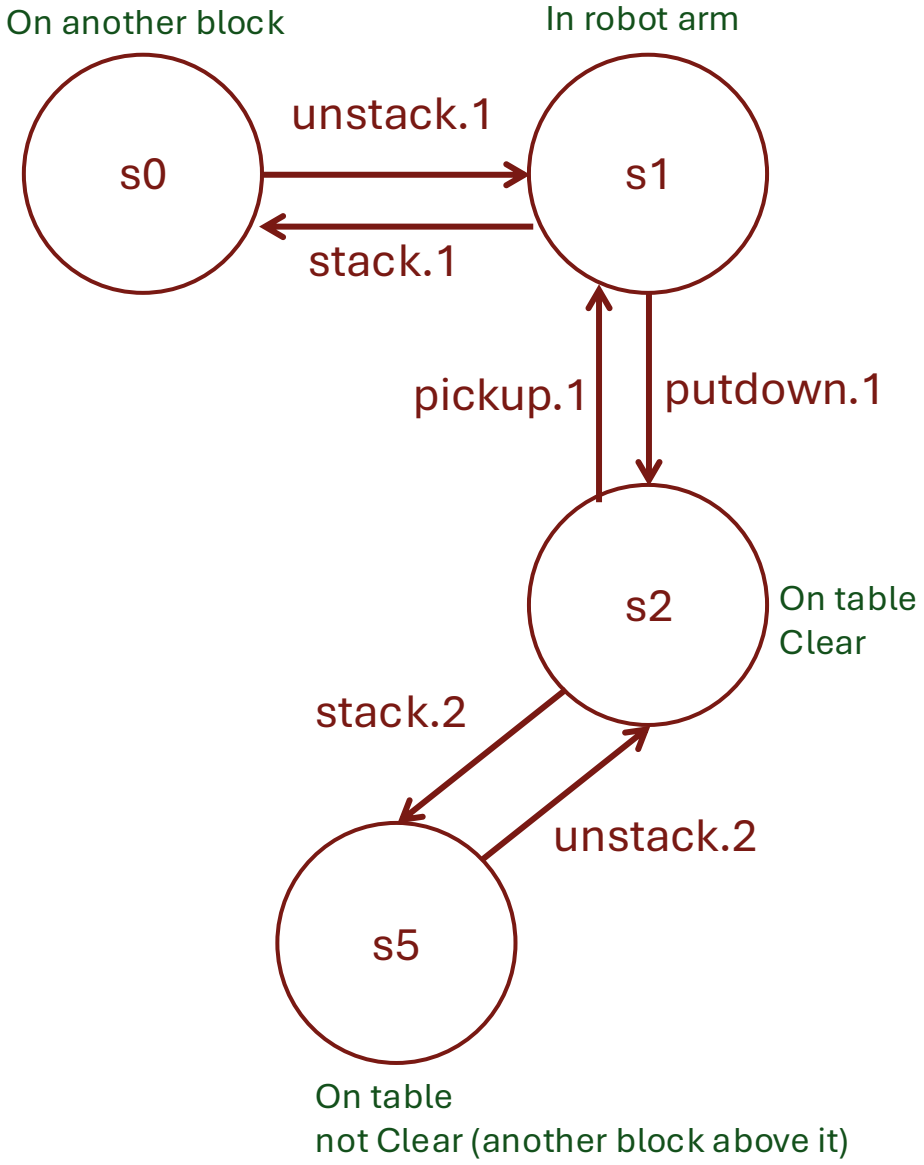


	Transition	Start	End
A	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	pickup.1	S_2	~~S_3~~=S_1
	stack.1	~~S_3~~ S_1	~~S_4~~=S_0
B	unstack.2	S_5	~~S_6~~=S_2
	pickup.1	~~S_6~~ S_2	~~S_7~~ S_3
	stack.1	S_3	~~S_4~~=S_0
	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	stack.2	S_2	~~S_7~~=S_5
C	stack.2	S_2	~~S_7~~=S_5
	unstack.2	S_5	S_2
	pickup.1	S_2	~~S_3~~=S_1
	putdown.1	S_1	S_2

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	pickup	C	-
10	putdown	C	-

Inducing FSMs



	Transition	Start	End
A	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	pickup.1	S_2	~~S_3~~=S_1
	stack.1	~~S_3~~ S_1	~~S_4~~=S_0
B	unstack.2	S_5	~~S_6~~=S_2
	pickup.1	~~S_6~~ S_2	~~S_7~~ S_3
	stack.1	S_3	~~S_4~~=S_0
	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	stack.2	S_2	~~S_7~~=S_5
C	stack.2	S_2	~~S_7~~=S_5
	unstack.2	S_5	S_2
	pickup.1	S_2	~~S_3~~=S_1
	putdown.1	S_1	S_2

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	pickup	C	-
10	putdown	C	-

Zero Analysis

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

- **Goal:** Detect implicit "background" objects,like the robot arm, that are never named in the trace but whose state constrains what can happen next.
- Treat every action as if it also acts on a single invisible "zero object" at position 0. Build its FSM using the same rules as Step 1.

Transition	Start	End
unstack.0		
putdown.0		
pickup.0		
stack.0		
unstack.0		
putdown.0		
pickup.0		
stack.0		
pickup.0		
putdown.0		

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	pickup	C	-
10	putdown	C	-

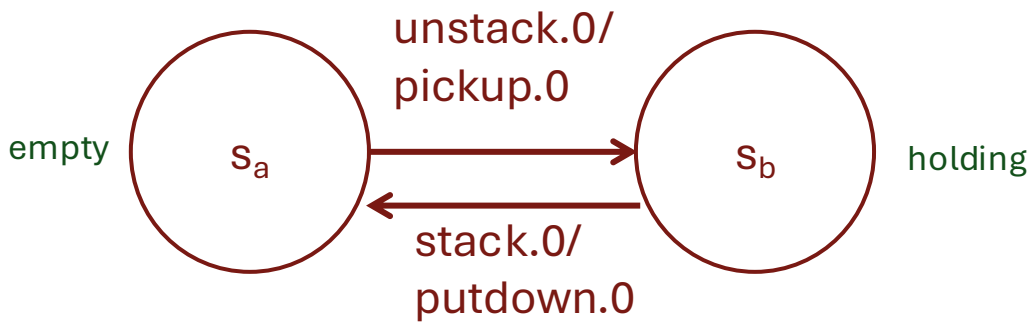
unstack.0 → putdown.0 → pickup.0 → stack.0 → unstack.0 → putdown.0 → pickup.0 → stack.0 → pickup.0 → putdown.0

Zero Analysis

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

- **Goal:** Detect implicit "background" objects,like the robot arm, that are never named in the trace but whose state constrains what can happen next.
- Treat every action as if it also acts on a single invisible "zero object" at position 0. Build its FSM using the same rules as Step 1.

Transition	Start	End
unstack.0		
putdown.0		
pickup.0		
stack.0		
unstack.0		
putdown.0		
pickup.0		
stack.0		
pickup.0		
putdown.0		



#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	pickup	C	-
10	putdown	C	-

Parameterization

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

- **Goal:** Discover associations between objects across sorts.
- E.g., when the arm is in state H1 (holding), which block is it holding?
- This requires linking the arm's FSM to the block sort's FSM via a shared parameter.
- 3 Steps:
  - Parameterize FSMs
  - Create and Merge Parameters
  - Remove Flawed Parameters.

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	pickup	C	-
10	putdown	C	-

Parameterize FSMs

- We have FSMs with states, but the states have no arguments
- State S_0 just says "this block is on another block." It does not say which block.

- Can we figure out which other object is associated with a state, purely from the trace?
  - Yes, by looking at what "bystander" objects appear alongside an object as it enters and exits a state.

The hypothesis structure

- A hypothesis is a claim about one specific state S:
  - "State S carries a parameter of sort G', where the incoming transition T_1 sets it from position k', and the outgoing transition T_2 reads it from position l'."

- Formally written as the tuple: ⟨S, T1, k, k', T2, l, l', G⟩
  - S = the state being parameterized
  - T_1 = the transition that enters S (the action and position of the main object)
  - k = position of the main object in action T_1
  - k' = position of the bystander in action T_1
  - T_2 = the transition that exits S
  - l = position of the main object in action T_2
  - l' = position of the bystander in action T_2
  - G = sort of the bystander

The hypothesis structure

Phase 1: Generating hypotheses

- For every object O, for every consecutive pair of transitions (T_1, T_2) in O's sequence:
  1. Look at every other argument position k' in action T_1 (not the position k that O occupies)
  2. Look at every other argument position l' in action T_2 (not the position l that O occupies)
  3. Find the objects at those positions in the actual trace
  4. If the object at k' in T_1 and the object at l' in T_2 are the same object, and they belong to the same sort G', then form the hypothesis

The hypothesis structure

- Phase 1: Generating hypotheses

  $\langle S, T_1, k, k', T_2, l, l', G \rangle$

1. Pair 1: (unstack.1, putdown.1)
   - In unstack(A, B): A at k=1. Bystander positions: k'=2, object = B
   - In putdown(A): A at l=1. Bystander positions: none
2. Pair 2: (putdown.1, pickup.1)
   - In putdown(A): A at k=1. Bystander positions: none
   - In pickup(A): A at l=1. Bystander positions: none
3. Pair 3: (pickup.1, stack.1)
   - In pickup(A): A at k=1. Bystander positions: none
   - In stack(A, B): A at l=1. Bystander at l'=2, object = B

	Transition	Start	End
A	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	pickup.1	S_2	~~S_3~~=S_1
	stack.1	~~S_3~~ S_1	~~S_4~~=S_0
B	unstack.2	S_5	~~S_6~~=S_2
	pickup.1	~~S_6~~ S_2	~~S_7~~ S_3
	stack.1	S_3	~~S_4~~=S_0
	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	stack.2	S_2	~~S_7~~=S_5
C	stack.2	S_2	~~S_7~~=S_5
	unstack.2	S_5	S_2
	pickup.1	S_2	~~S_3~~=S_1
	putdown.1	S_1	S_2

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	pickup	C	-
10	putdown	C	-

The hypothesis structure

- Phase 1: Generating hypotheses

  ⟨S, T_1, k, k', T_2, l, l', G⟩

1. Pair 1: (unstack.2, pickup.1)
2. Pair 2: (pickup.1, stack.1)
3. Pair 3: (stack.1, unstack.1)
   - In stack(B, C): B at k=1. Bystander at k'=2, object = C, sort = Sort1
   - In unstack(B, C): B at l=1. Bystander at l'=2, object = C, sort = Sort1
4. Pair 4: (unstack.1, putdown.1)
5. Pair 5: (putdown.1, stack.2)

	Transition	Start	End
A	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	pickup.1	S_2	~~S_3~~=S_1
	stack.1	~~S_3~~ S_1	~~S_4~~=S_0
B	unstack.2	S_5	~~S_6~~=S_2
	pickup.1	~~S_6~~ S_2	~~S_7~~ S_3
	stack.1	S_3	~~S_4~~=S_0
	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	stack.2	S_2	~~S_7~~=S_5
C	stack.2	S_2	~~S_7~~=S_5
	unstack.2	S_5	S_2
	pickup.1	S_2	~~S_3~~=S_1
	putdown.1	S_1	S_2

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	pickup	C	-
10	putdown	C	-

The hypothesis structure

- Phase 1: Generating hypotheses
  ⟨S, T_1, k, k', T_2, l, l', G⟩

1. Pair 1: (stack.2, unstack.2)
   - In stack(B, C): B at k=2. Bystander at k'=1, object = B, sort = Sort1
   - In unstack(B, C): B at l=2. Bystander at l'=1, object = B, sort = Sort1
2. Pair 2: (unstack.2, pickup.1)
3. Pair 4: (pickup.1, putdown.1)

	Transition	Start	End
A	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	pickup.1	S_2	~~S_3~~=S_1
	stack.1	~~S_3~~ S_1	~~S_4~~=S_0
B	unstack.2	S_5	~~S_6~~=S_2
	pickup.1	~~S_6~~ S_2	~~S_7~~ S_3
	stack.1	S_3	~~S_4~~=S_0
	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	stack.2	S_2	~~S_7~~=S_5
C	stack.2	S_2	~~S_7~~=S_5
	unstack.2	S_5	S_2
	pickup.1	S_2	~~S_3~~=S_1
	putdown.1	S_1	S_2

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	pickup	C	-
10	putdown	C	-

The hypothesis structure

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

$\langle S, T_1, k, k', T_2, I, I', G \rangle$

- Phase 1 summary:

1. H_1: $\langle S_0$, stack.1, k=1, k'=2, unstack.1, I=1, I'=2, Sort1$\rangle$
2. H_2: $\langle S_5$, stack.2, k=2, k'=1, unstack.2, I=2, I'=1, Sort1$\rangle$

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	pickup	C	-
10	putdown	C	-

The hypothesis structure

- Testing H_1: ⟨S_0, stack.1, k=1, k'=2, unstack.1, l=1, l'=2, Sort1⟩

- Scan the entire trace for every consecutive pair where stack.1 is immediately followed (for the same object) by unstack.1.
  - stack(B, C): bystander at pos 2 = C
  - unstack(B, C): bystander at pos 2 = C

- H_1 is valid

	Transition	Start	End
A	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	pickup.1	S_2	~~S_3~~=S_1
	stack.1	~~S_3~~ S_1	~~S_4~~=S_0
B	unstack.2	S_5	~~S_6~~=S_2
	pickup.1	~~S_6~~ S_2	~~S_7~~ S_3
	stack.1	S_3	~~S_4~~=S_0
	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	stack.2	S_2	~~S_7~~=S_5
C	stack.2	S_2	~~S_7~~=S_5
	unstack.2	S_5	S_2
	pickup.1	S_2	~~S_3~~=S_1
	putdown.1	S_1	S_2

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	pickup	C	-
10	putdown	C	-

The hypothesis structure

- Testing H_2: ⟨S_5, stack.2, k=2, k'=1, unstack.2, l=2, l'=1, Sort1⟩

- Scan the entire trace for every consecutive pair where stack.2 is immediately followed (for the same object) by unstack.2.
  - stack(B, C): bystander at pos 1 = B
  - unstack(B, C): bystander at pos 1 = B

- H_2 is valid

	Transition	Start	End
A	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	pickup.1	S_2	✗ S_3=S_1
	stack.1	✗ S_3 S_1	✗ S_4=S_0
B	unstack.2	S_5	✗ S_6=S_2
	pickup.1	✗ S_6 S_2	✗ S_7 S_3
	stack.1	S_3	✗ S_4=S_0
	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	stack.2	S_2	✗ S_7=S_5
C	stack.2	S_2	✗ S_7=S_5
	unstack.2	S_5	S_2
	pickup.1	S_2	✗ S_3=S_1
	putdown.1	S_1	S_2

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	pickup	C	-
10	putdown	C	-

The hypothesis structure

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

Final result of Step 3: Both hypotheses survive. The parameterized states are:

- **S_0(?x, ?y)** — block ?x is on block ?y
  - Parameter set when stack(?x, ?y) fires: ?y is at position 2
  - Parameter read when unstack(?x, ?y) fires: ?y is at position 2
- **S_5(?x, ?y)** — block ?x has block ?y on top of it
  - Parameter set when stack(?y, ?x) fires: ?y is at position 1
  - Parameter read when unstack(?y, ?x) fires: ?y is at position 1

Why are we doing this?

By default, each state has one parameter, as it is capturing some property of the main object. In this step we try to learn if there is a bystander object too, whose property is consistently captured by this state.

Create and Merge Parameters

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

Assume the trace was richer

- After Step 3

1. H_1: $\langle S_0$, stack.1, k=1, k'=2, unstack.1, l=1, l'=2, Sort1$\rangle$
2. H_3: $\langle S_0$, stack.1, k=1, k'=2, unstack.1, l=1, l'=2, Sort1$\rangle$
3. H_2: $\langle S_5$, stack.2, k=2, k'=1, unstack.2, l=2, l'=1, Sort1$\rangle$
4. H_4: $\langle S_5$, stack.2, k=2, k'=1, unstack.2, l=2, l'=1, Sort1$\rangle$

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	stack	C	A
10	unstack	C	A

Create and Merge Parameters

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

Assume the trace was richer

- After Step 3

1. H_1: $\langle S_0$, stack.1, k=1, k'=2, unstack.1, l=1, l'=2, Sort1$\rangle$
2. H_2: $\langle S_5$, stack.2, k=2, k'=1, unstack.2, l=2, l'=1, Sort1$\rangle$

#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
9.	stack	C	A
10	unstack	C	A

Remove Flawed Parameters

- S_0 would have two incoming transitions:
  - `stack.1`
  - place.1

- If a block can reach S_0 via place.1 without establishing which block it is resting on, then S_0 cannot reliably carry that information. The parameter would be undefined for blocks that arrived via place.1, making it useless and potentially incorrect.

	Transition	Start	End
A	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	pickup.1	S_2	~~S_3~~=S_1
	stack.1	~~S_3~~ S_1	~~S_4~~=S_0
	unstack.2	S_5	~~S_6~~=S_2
B	place.1	S_2	S_0
	pickup.1	~~S_6~~ S_2	~~S_7~~ S_3
	stack.1	S_3	~~S_4~~=S_0
	unstack.1	S_0	S_1
	putdown.1	S_1	S_2
	stack.2	S_2	~~S_7~~=S_5
C	stack.2	S_2	~~S_7~~=S_5
	unstack.2	S_5	~~S_2~~
	pickup.1	S_2	S_3=S_1
	putdown.1	S_1	S_2

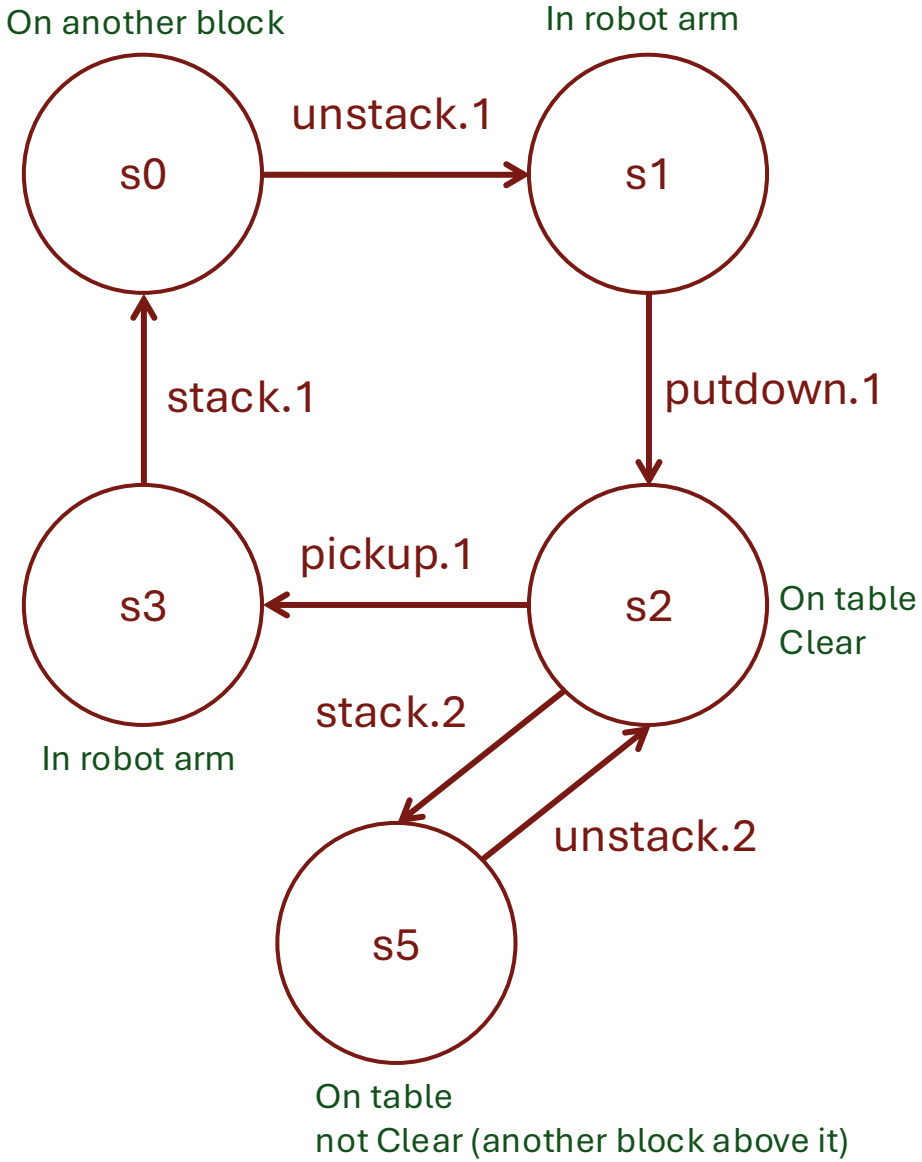
0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

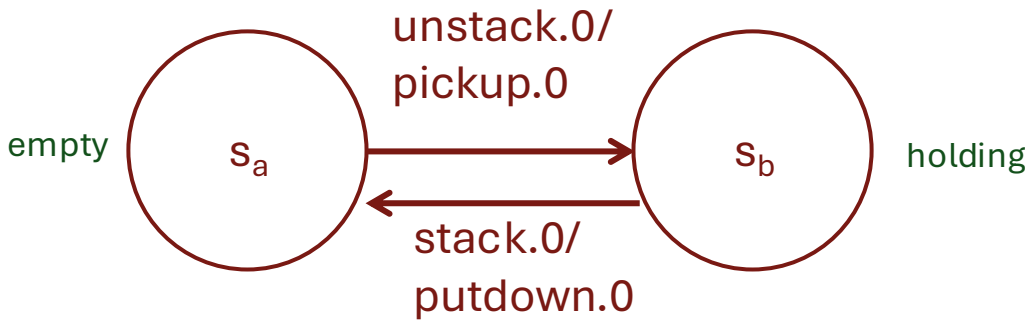
#	Action	arg1	arg2
1.	unstack	A	B
2.	putdown	A	-
3.	pickup	B	-
4.	stack	B	C
5.	unstack	B	C
6.	putdown	B	-
7.	pickup	A	-
8.	stack	A	B
11	place	A	-

Static preconditions

- Limitation of LOCM: Have to be input manually

PDDL Schema





0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

pickup(?x):

- **From Sort 1 FSM:** transition pickup.1 goes $S_2 \rightarrow S_3$
  - precondition: sort1_s2(?x) : block is on table
  - delete: sort1_s2(?x)
  - add: sort1_s3(?x) : block is in arm
- **From Sort 0 FSM:** transition pickup.0 goes $S_a \rightarrow S_b(?x)$
  - precondition: sort0_sa : arm is empty
  - delete: sort0_sa
  - add: sort0_sb(?x): arm is holding ?x

PDDL Schema

0	Sort Formation
1	Induce FSMs
2	Zero Analysis
3	Parameterize FSMs
4	Create & Merge Params
5	Remove Flawed Params
6	Static Preconditions
7	Output PDDL Schema

pickup(?x):

- **From Sort 1 FSM:** transition pickup.1 goes $S_2 \rightarrow S_3$
  - precondition: sort1_s2(?x) : block is on table
  - delete: sort1_s2(?x)
  - add: sort1_s3(?x) : block is in arm
- **From Sort 0 FSM:** transition pickup.0 goes $S_a \rightarrow S_b(?x)$
  - precondition: sort0_sa : arm is empty
  - delete: sort0_sa
  - add: sort0_sb(?x): arm is holding ?x

```
(:action pickup (?x)
 :precondition (and (sort1_s2 ?x)
                    (sort0_sa))
 :effect (and (sort1_s3 ?x)
              (sort0_sb ?x)
              (not (sort1_s2 ?x))
              (not (sort0_sa))))
```