Lecture 20

CS 6260 Automated Planning and Learning

Learning Object-Centred Models (LOCM)

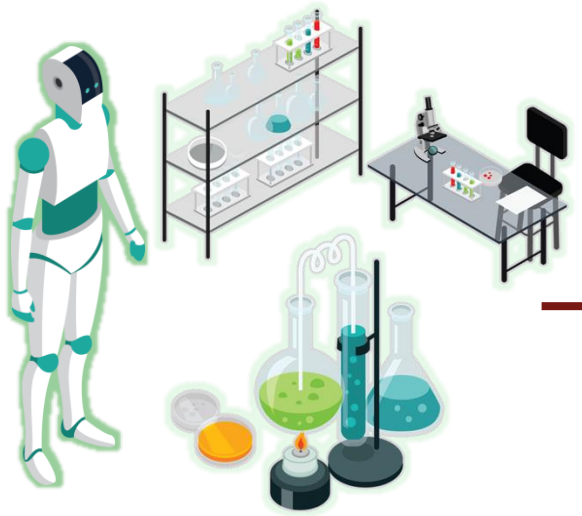
Department of Computer Science and Engineering

Indian Institute of Technology Madras

Recap: Learning PDDL Models

Reference: https://domain-learning.github.io/

Learning Action Models from Passive Observations



$\langle s_0, a_1, s_1 \rangle$
$\langle s_1, a_2, s_2 \rangle$
$\vdots$
$\langle s_{n-1}, a_n, s_n \rangle$

[Input]

Learner

```
(:action pick-up-beaker
   :parameters (?x - beaker)
      :precondition (and (ontable ?x)
         (not-in-use ?x)
         (handempty))
      :effect (and (not (ontable ?x))
          (not (handempty))
          (holding ?x)))

(:action put-on-shelf
   :parameters (?x - beaker)
      :precondition (holding ?x)
      :effect (and (not (holding ?x))
          (handempty)
          (onshelf ?x)))
```

[PDDL Example]

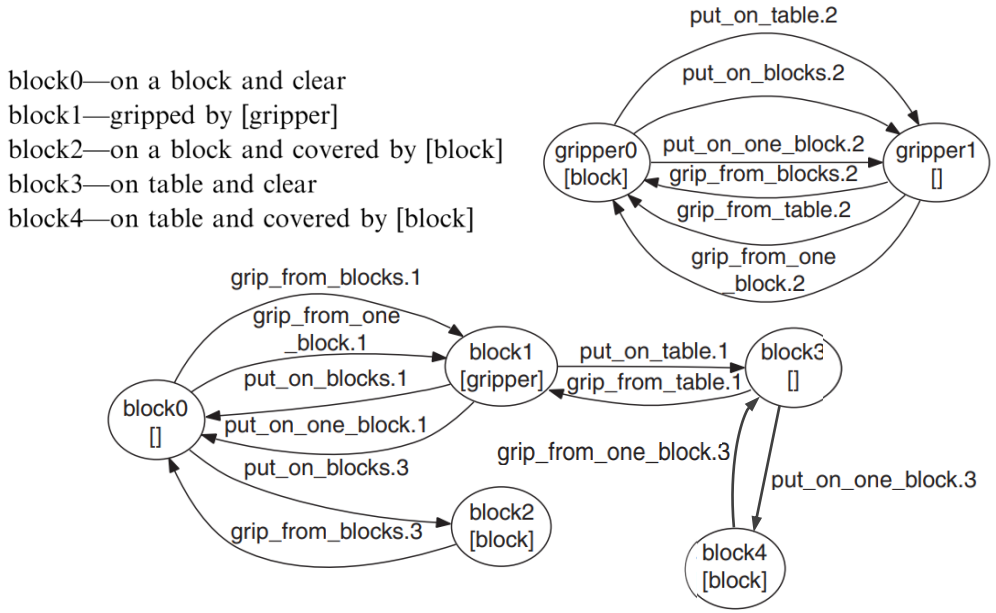
What kind of approaches these learners use?

Finite State Machine based Learners

- For each object type create a finite state machine.
- Create PDDL by combining them.

$\langle s_0, a_1, s_1 \rangle$
$\langle s_1, a_2, s_2 \rangle$
$\vdots$
$\langle s_{n-1}, a_n, s_n \rangle$

[Input]




```
(:action pick-up-beaker
   :parameters (?x - beaker)
      :precondition (and (ontable ?x)
         (not-in-use ?x)
         (handempty))
      :effect (and (not (ontable ?x))
          (not (handempty))
          (holding ?x)))

(:action put-on-shelf
   :parameters (?x - beaker)
      :precondition (holding ?x)
      :effect (and (not (holding ?x))
          (handempty)
          (onshelf ?x)))
```

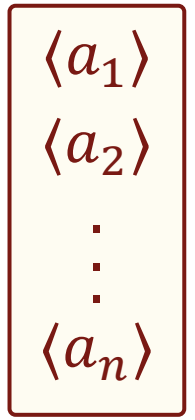
[PDDL Example]

LOCM, LOCM2 - Cresswell et al. (ICAPS'09 , ICAPS'11, Know. Engg. Rev.'13), LOP - Gregory et al. (ICAPS'15), NLOCM - Gregory et al. (ICAPS'16)

Learning Object-Centred Models (LOCM)

- Input: Action Sequence
  - List of action names with object arguments

- What is not part of the input?
  - Any description of states before, during, or after actions
  - Any predicates or fluents
  - The initial state or goal state
  - Any type/sort information about objects
  - Any information about what the actions mean or how they work
  - Any knowledge of agent rationality or plan optimality



[Input]

LOCM: High Level Approach

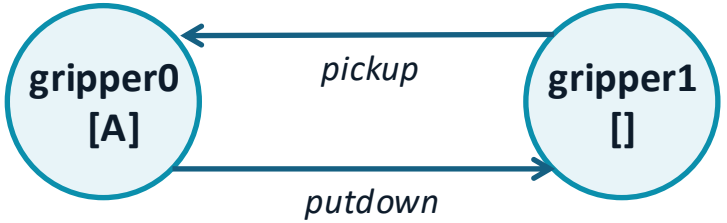
- Takes a single action sequence as the only input
- Identifies unique object **sorts** (types) in the domain
- Builds Finite State Machines (FSMs) for each sort
- Augments FSMs with inter-object parameter relationships
- Outputs lifted PDDL action models

7 Steps + Sort Formation

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

Finite State Machines (FSMs) for LOCM

- Each object sort gets its own FSM
- States represent the condition of an object
- Actions cause transitions between states
- Consecutive actions must share a state
- Same action always produces the same start → end transition



Objects transition between states as actions are applied.
LOCM reconstructs these FSMs purely from the action sequence.

Parameterization: Learning Object Relationships

Step 3

Form Hypotheses

If two consecutive transitions share an argument of the same sort, hypothesize that their shared state has a parametric association to that sort.

Step 4

Merge Parameters

Check all pairs of hypotheses. When the same incoming or outgoing action references the same position, unify the parameter variables using set merging.

Step 5

Remove Flawed Params

Remove parameters not set by all incoming transitions — these are indeterminate and would produce incorrect action models.

Result: Fully parameterized FSMs capturing both object states and cross-sort relationships

LOCM on Blockworld

- Actions:
  - pickup(?x)
  - putdown(?x)
  - stack(?x, ?y)
  - unstack(?x, ?y)

- Objects:
  - A, B, C

- Input Trace:
  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)

Sort Formation

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:** Group objects into sorts (types) by observing which objects share argument positions.
- **Rule:** Any two objects that ever appear in the same argument position of the same action must belong to the same sort.

#	Action	arg1	arg2	Sort
1.	unstack	A	B	A at arg1 → start $Sort_1 = \{A\}$. B at arg2 → start $Sort_2 = \{B\}$
2.	putdown	A	-	A at arg1 of putdown → A already in $Sort_1$. No change.
3.	pickup	B	-	B at arg1 of pickup. B is currently in $Sort_2$.
4.	stack	B	C	B at arg1 of stack, and is already in $Sort_1$, C at arg2 → start $Sort3 = \{C\}$
5.	unstack	B	C	B and A should fall in same sort as they are arg1 of unstack. Similarly, B and C should fall in the same sort as they are arg2 of unstack. So, B and C must join $Sort_1$. **$Sort_1 = \{A,B,C\}$.**
6.	putdown	B	-	B already in $Sort_1$.
7.	pickup	A	-	A already in $Sort_1$.
8.	stack	A	B	A and B already in $Sort_1$.

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

- **Goal:** For each sort, build an FSM by tracking each object's transition sequence and applying the continuity rule.
- A *transition* `action.k` = the state change caused by an object appearing at argument position k of that action
- **Continuity:** $\text{end}(\text{transition}_i) = \text{start}(\text{transition}_{i+1})$ for consecutive transitions on the same object
- **1-1 rule:** the same `action.k` always has the same start state and end state across all objects

#	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

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

- A *transition* `action.k` = the state change caused by an object appearing at argument position k of that action
- **Continuity:** end(transition$_i$) = start(transition$_{i+1}$) for consecutive transitions on the same object
- **1-1 rule:** the same `action.k` always has the same start state and end state across all objects

#	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

Object A appears at:

- Step 1: `unstack.1` (arg1 of unstack)
- Step 2: `putdown.1` (arg1 of putdown)
- Step 7: `pickup.1` (arg1 of pickup)
- Step 8: `stack.1` (arg1 of stack)

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

- A *transition* `action.k` = the state change caused by an object appearing at argument position k of that action
- **Continuity:** end(transition$_i$) = start(transition$_{i+1}$) for consecutive transitions on the same object
- **1-1 rule:** the same `action.k` always has the same start state and end state across all objects

A

Transition	Start	End	Reason
`unstack.1`	S_0	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	Continuity: start = S_3. Assign fresh end S_4

#	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

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

- A *transition* `action.k` = the state change caused by an object appearing at argument position k of that action
- **Continuity:** end(transition$_i$) = start(transition$_{i+1}$) for consecutive transitions on the same object
- **1-1 rule:** the same `action.k` always has the same start state and end state across all objects

A

Transition	Start	End	Reason
`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	Continuity: start = S_3. Assign fresh end S_4
`unstack.2`	S_5	S_6	first time seen, assign $S_5 \rightarrow S_6$

#	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

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

- A *transition* `action.k` = the state change caused by an object appearing at argument position k of that action
- **Continuity:** end(transition$_i$) = start(transition$_{i+1}$) for consecutive transitions on the same object
- **1-1 rule:** the same `action.k` always has the same start state and end state across all objects

A

Transition	Start	End	Reason
`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	Continuity: start = S_3. Assign fresh end S_4
`unstack.2`	S_5	S_6	first time seen, assign $S_5 \rightarrow S_6$
`pickup.1`	~~S_6~~ S_2	~~S_7~~ S_3	Using 1-1 rule

#	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

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

- A *transition* `action.k` = the state change caused by an object appearing at argument position k of that action
- **Continuity:** end(transition$_i$) = start(transition$_{i+1}$) for consecutive transitions on the same object
- **1-1 rule:** the same `action.k` always has the same start state and end state across all objects

A

Transition	Start	End	Reason
`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	Continuity: start = S_3. Assign fresh end S_4
`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

#	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

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

- A *transition* `action.k` = the state change caused by an object appearing at argument position k of that action
- **Continuity:** end(transition$_i$) = start(transition$_{i+1}$) for consecutive transitions on the same object
- **1-1 rule:** the same `action.k` always has the same start state and end state across all objects

A

Transition	Start	End	Reason
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	Continuity: start = S_3. Assign fresh end S_4
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	

#	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

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

- A *transition* `action.k` = the state change caused by an object appearing at argument position k of that action
- **Continuity:** end(transition$_i$) = start(transition$_{i+1}$) for consecutive transitions on the same object
- **1-1 rule:** the same `action.k` always has the same start state and end state across all objects

A

Transition	Start	End	Reason
`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	Continuity: start = S_3. Assign fresh end S_4
`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	
`unstack.1`	S_0	S_1	

#	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

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

- A *transition* `action.k` = the state change caused by an object appearing at argument position k of that action
- **Continuity:** end(transition$_i$) = start(transition$_{i+1}$) for consecutive transitions on the same object
- **1-1 rule:** the same `action.k` always has the same start state and end state across all objects

A

Transition	Start	End	Reason
`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
`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	

#	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

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

- A *transition* `action.k` = the state change caused by an object appearing at argument position k of that action
- **Continuity:** end(transition$_i$) = start(transition$_{i+1}$) for consecutive transitions on the same object
- **1-1 rule:** the same `action.k` always has the same start state and end state across all objects

A

Transition	Start	End	Reason
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
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	

#	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

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	

A

Transition	Start	End	Reason
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
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	

#	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

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	

#	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

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 $S_0 \rightarrow S_1$
A	putdown.1	S_1	S_2	Continuity: start must = end of prev (S_1). Assign fresh end S_2
A	pickup.1	S_2	S_3	Continuity: start = S_2. Assign fresh end S_3
A	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 $S_5 \rightarrow S_6$
B	pickup.1	~~S_6~~ S_2	~~S_7~~ S_3	Using 1-1 rule
B	stack.1	S_3	~~S_4~~=S_0	
B	unstack.1	S_0	S_1	
B	putdown.1	S_1	S_2	
B	stack.2	S_2	S_7	
	stack.2	S_2	S_7	

#	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

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	
C	stack.2	S_2	S_7	
	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

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