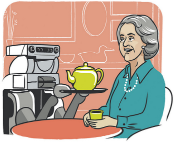
Learning Hierarchical Goal-Oriented Tasks from Situated Interactive Instruction

Shiwali Mohan, John E. Laird

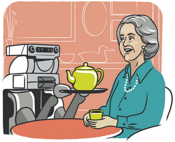
Computer Science and Engineering
University of Michigan, Ann Arbor

June 18th, 2014

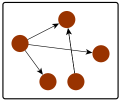
Interactive Task Learning



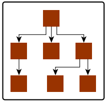
Interactive Task Learning



characteristics

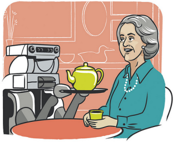


relational goal structure

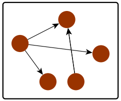


hierarchical decomposition

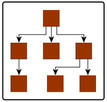
Interactive Task Learning



characteristics



relational goal structure

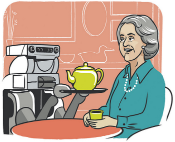


hierarchical decomposition

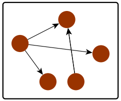
acquire

- what?
- how?
- when?

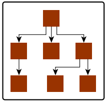
Interactive Task Learning



characteristics



relational goal structure



hierarchical decomposition

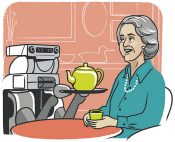
desiderata

- multi-task learning
- assisted transfer
- fast generalization
- distributed initiative

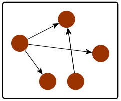
acquire

- what?
- how?
- when?

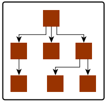
Interactive Task Learning



characteristics



relational goal structure



hierarchical decomposition

acquire

- what?
- how?
- when?

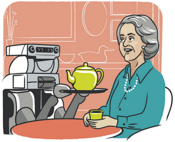
desiderata

- multi-task learning
- assisted transfer
- fast generalization
- distributed initiative

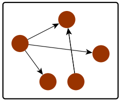
approach

- composable, hierarchical representations
- knowledge-rich machine learning - EBL
- demo

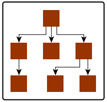
Interactive Task Learning



characteristics



relational goal structure



hierarchical decomposition

acquire

- what?
- how?
- when?

desiderata

- multi-task learning
- assisted transfer
- fast generalization
- distributed initiative

approach

- soar state stack
  + operators
- chunking
  + selection
- demo

Task Representation

For the task `store`:

What?

- parameters
  Store the green cylinder.
  `store([o], pantry, in([o],pantry))`
- subtasks
  `store`: `open`, `move [pick-up, put-down]`, `close`
- goal
  `in(O2,pantry)` ∧ `closed(pantry)`

How?

- policy
  if `[state,task]` then `execute([subtask])`
- model
  if `[state,task]` then `[next-state]`

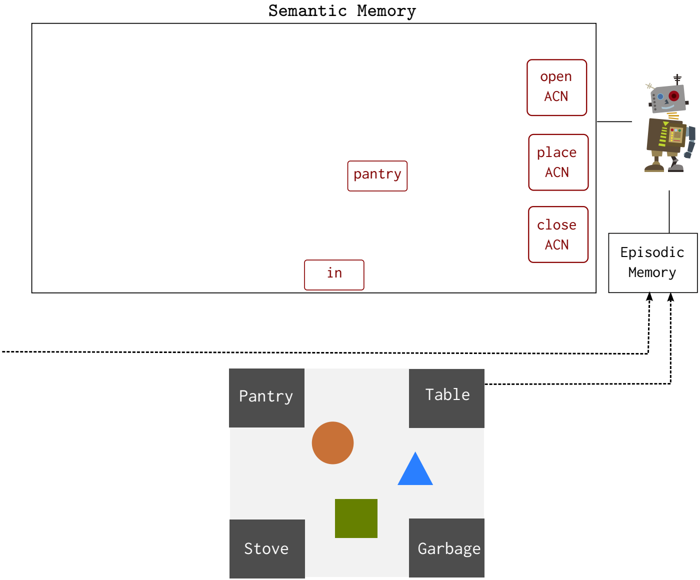
When?

- availability
  if `[state]` then `available(store)`
- termination
  if `[state]` then `terminate(store)`

Interactive Example Execution

Interaction trace

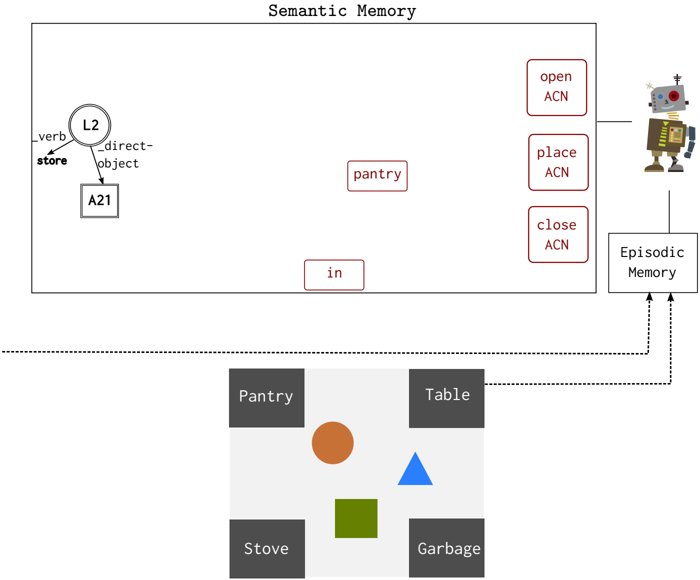
Instructor: Store the green rectangle.



Interactive Example Execution

Interaction trace

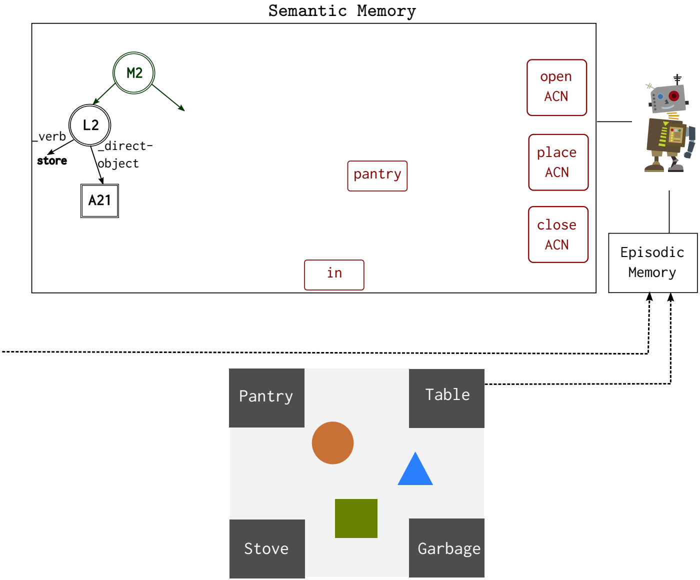
Instructor: Store the green rectangle.



Interactive Example Execution

Interaction trace

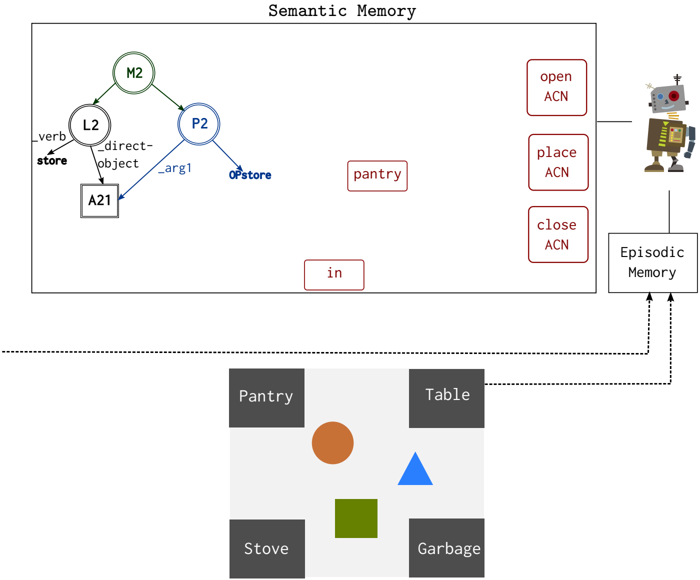
Instructor: Store the green rectangle.



Interactive Example Execution

Interaction trace

Instructor: Store the green rectangle.

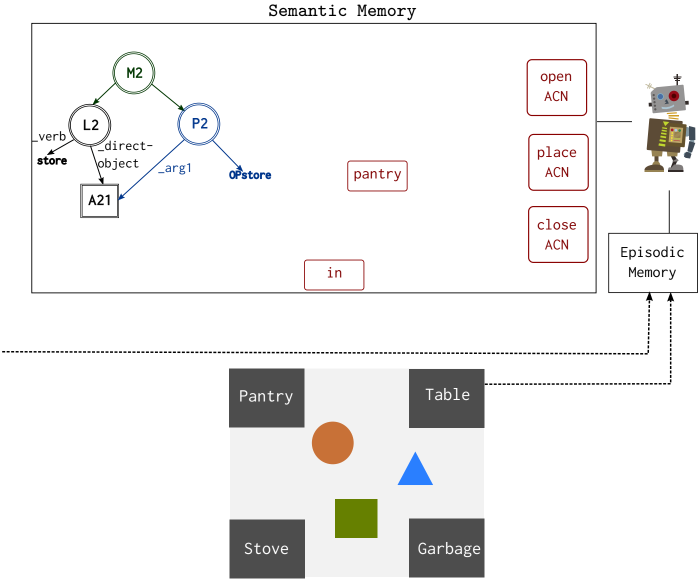


Interactive Example Execution

Interaction trace

Instructor: Store the green rectangle.

Agent: What is the goal of the action?



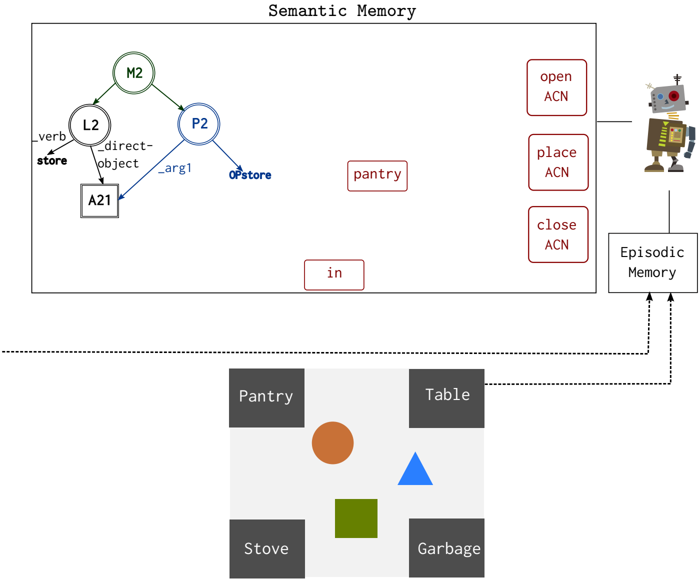
Interactive Example Execution

Interaction trace

Instructor: Store the green rectangle.

Agent: What is the goal of the action?

Instructor: The goal is the green rectangle in the pantry and the pantry closed.



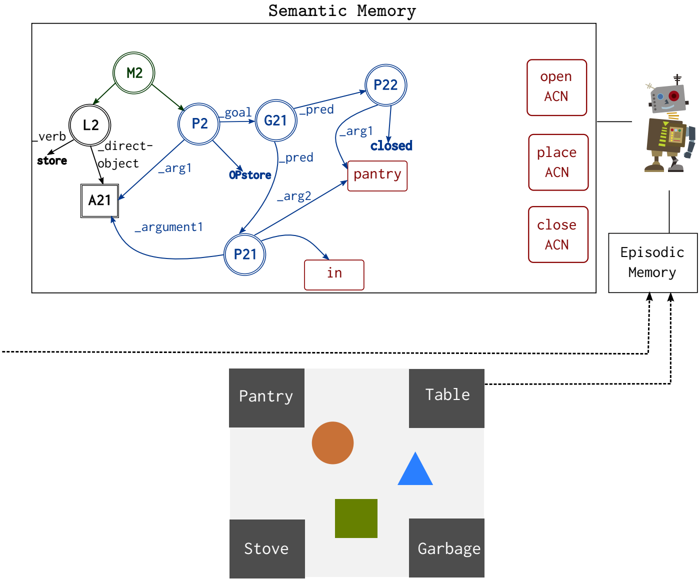
Interactive Example Execution

Interaction trace

Instructor: Store the green rectangle.

Agent: What is the goal of the action?

Instructor: The goal is the green rectangle in the pantry and the pantry closed.



Interactive Example Execution

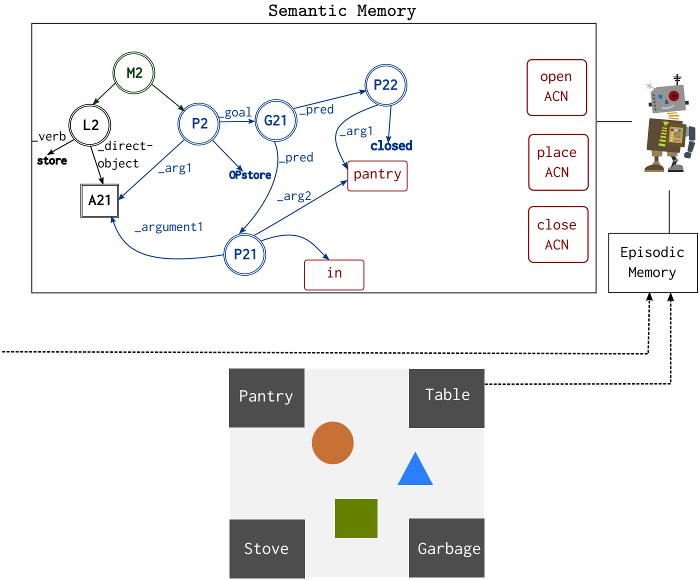
Interaction trace

Instructor: Store the green rectangle.

Agent: What is the goal of the action?

Instructor: The goal is the green rectangle in the pantry and the pantry closed.

Agent: Which action should I take?



Interactive Example Execution

Interaction trace

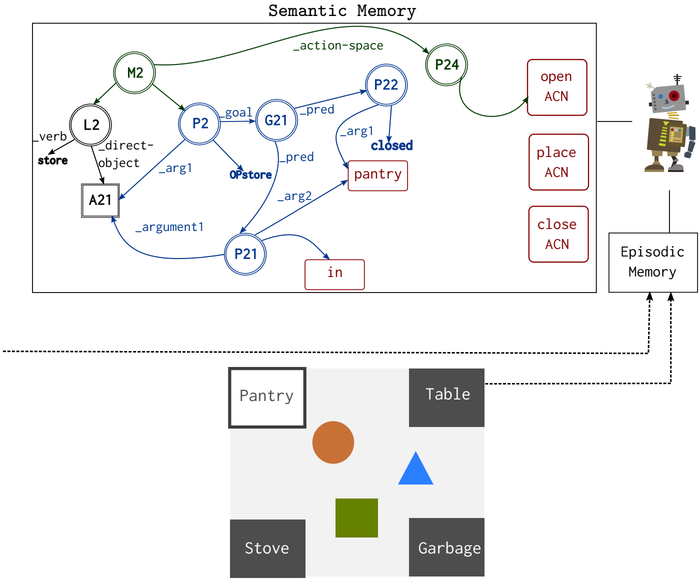
Instructor: Store the green rectangle.

Agent: What is the goal of the action?

Instructor: The goal is the green rectangle in the pantry and the pantry closed.

Agent: Which action should I take?

Instructor: Open the pantry.



Interactive Example Execution

Interaction trace

Instructor: Store the green rectangle.

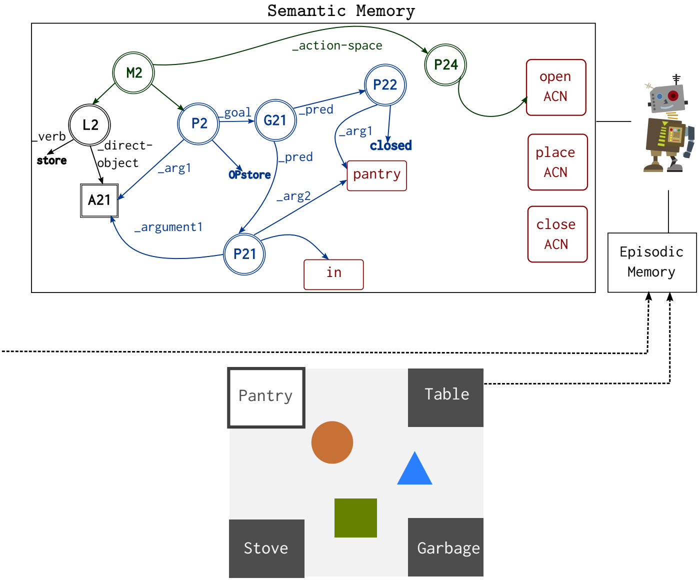
Agent: What is the goal of the action?

Instructor: The goal is the green rectangle in the pantry and the pantry closed.

Agent: Which action should I take?

Instructor: Open the pantry.

Agent: Which action should I take?



Interactive Example Execution

Interaction trace

Instructor: Store the green rectangle.

Agent: What is the goal of the action?

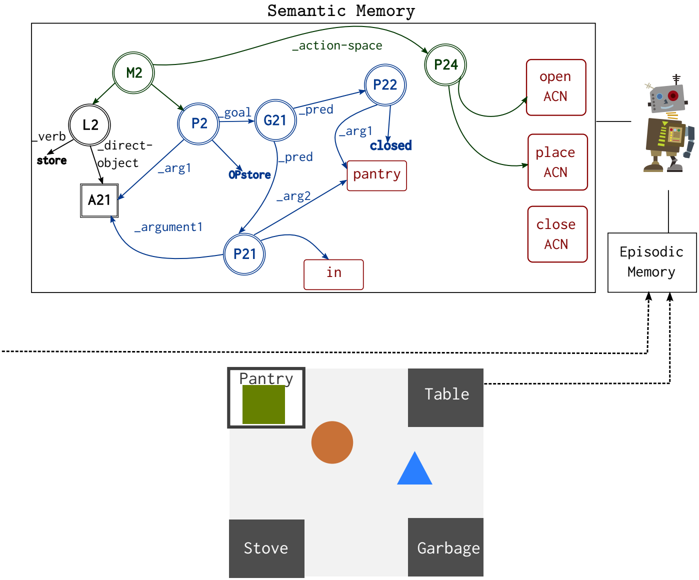
Instructor: The goal is the green rectangle in the pantry and the pantry closed.

Agent: Which action should I take?

Instructor: Open the pantry.

Agent: Which action should I take?

Instructor: Place the object in the pantry.



Interactive Example Execution

Interaction trace

Instructor: Store the green rectangle.

Agent: What is the goal of the action?

Instructor: The goal is the green rectangle in the pantry and the pantry closed.

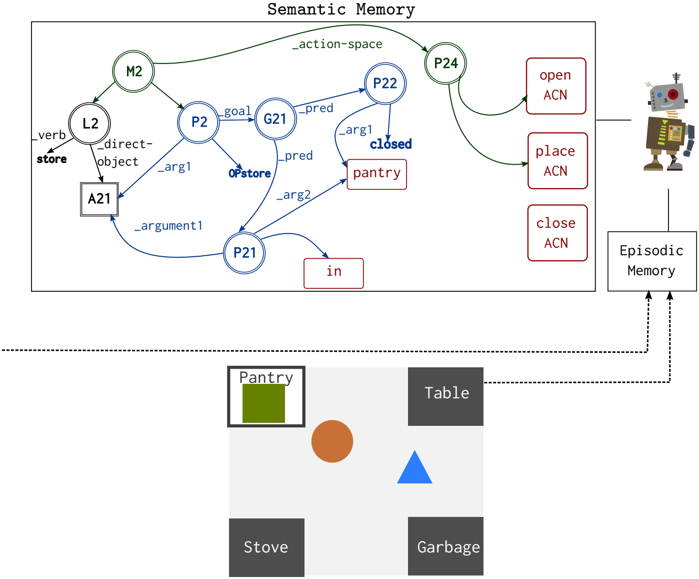
Agent: Which action should I take?

Instructor: Open the pantry.

Agent: Which action should I take?

Instructor: Place the object in the pantry.

Agent: OK.



Interactive Example Execution

Interaction trace

Instructor: Store the green rectangle.

Agent: What is the goal of the action?

Instructor: The goal is the green rectangle in the pantry and the pantry closed.

Agent: Which action should I take?

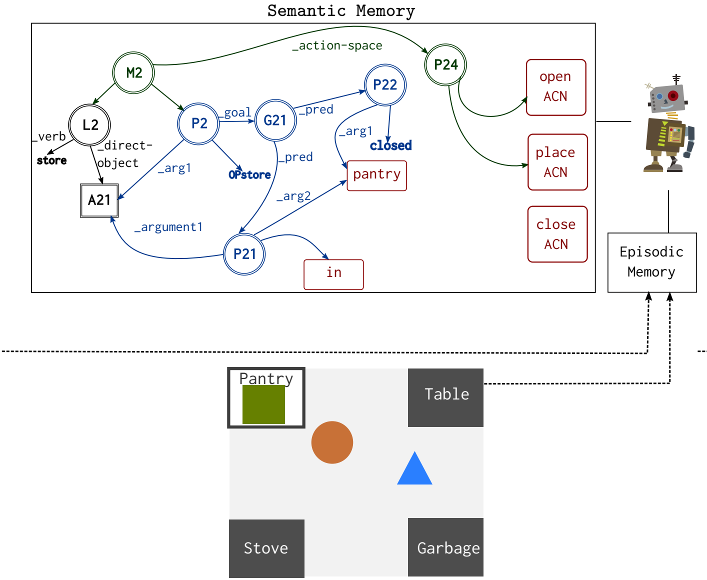
Instructor: Open the pantry.

Agent: Which action should I take?

Instructor: Place the object in the pantry.

Agent: OK.

Agent: `explores action space`



Interactive Example Execution

Interaction trace

Instructor: Store the green rectangle.

Agent: What is the goal of the action?

Instructor: The goal is the green rectangle in the pantry and the pantry closed.

Agent: Which action should I take?

Instructor: Open the pantry.

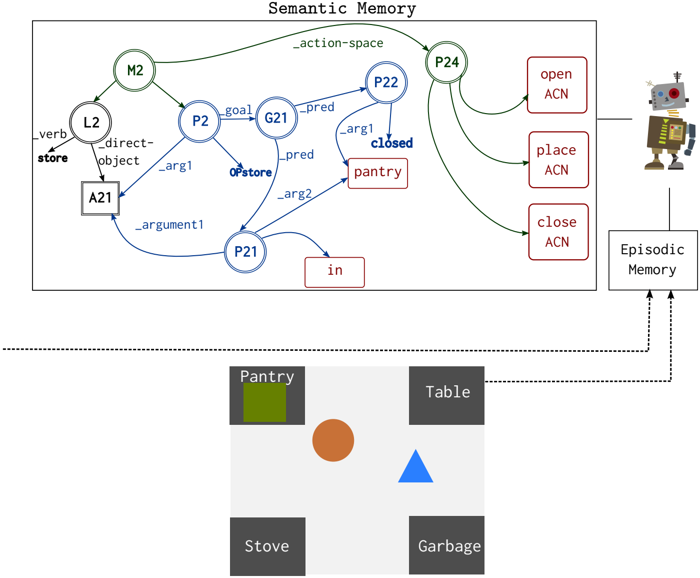
Agent: Which action should I take?

Instructor: Place the object in the pantry.

Agent: OK.

Agent: `explores action space`

Agent: `closes the pantry`



Interactive Example Execution

Interaction trace

Instructor: Store the green rectangle.

Agent: What is the goal of the action?

Instructor: The goal is the green rectangle in the pantry and the pantry closed.

Agent: Which action should I take?

Instructor: Open the pantry.

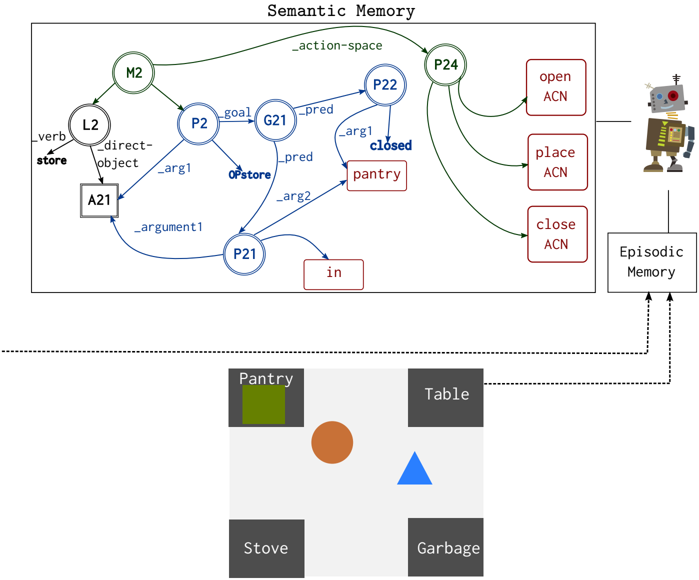
Agent: Which action should I take?

Instructor: Place the object in the pantry.

Agent: OK.

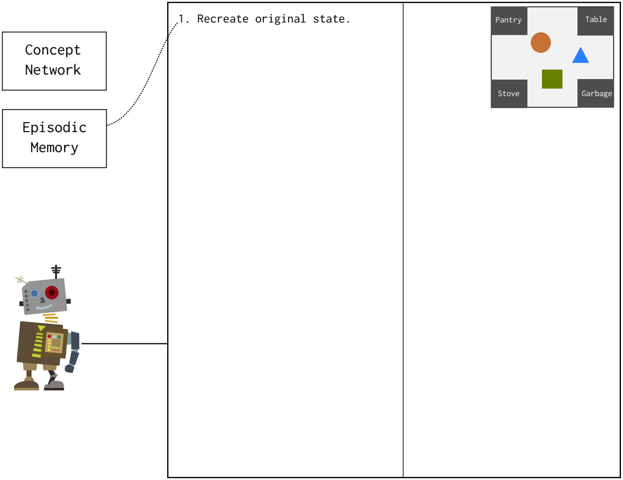
Agent: `explores action space`

Agent: `closes the pantry`



Retrospective EBL

Retrospective EBL

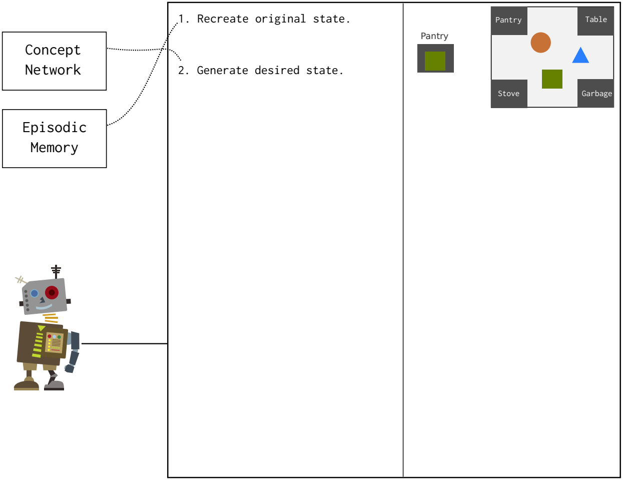


Concept Network

Episodic Memory

1. Recreate original state.

Retrospective EBL



Concept Network

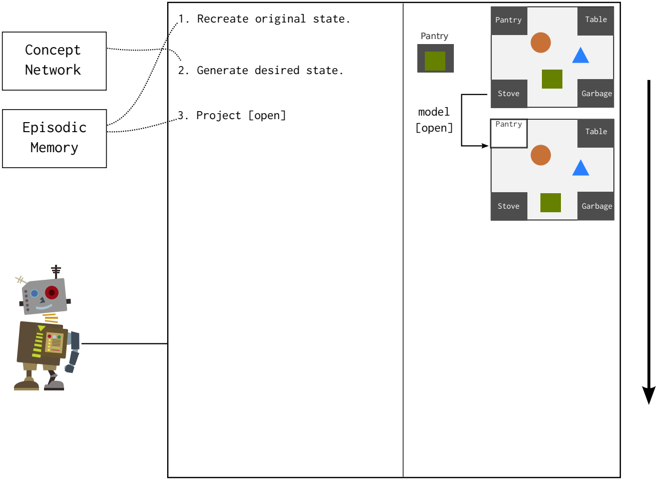
Episodic Memory

1. Recreate original state.

2. Generate desired state.

terminate store:
If store([x]) and
IN([x],PANTRY) and
CLOSED(PANTRY)
-->
terminate store[x]

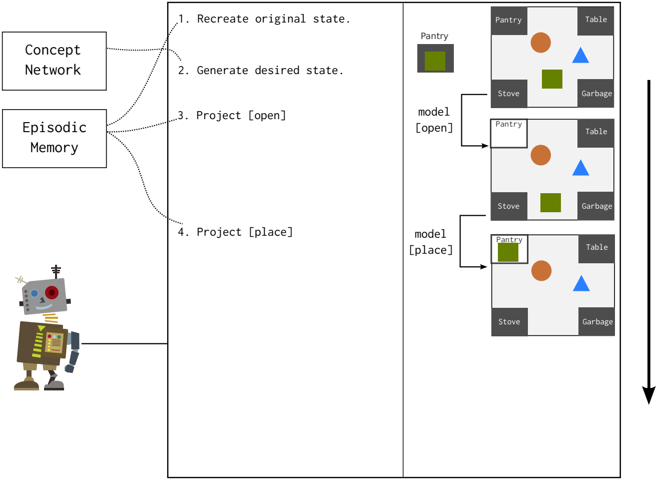
Retrospective EBL



1. Recreate original state.
2. Generate desired state.
3. Project [open]

terminate store:
If `store([x])` and
`IN([x],PANTRY)` and
`CLOSED(PANTRY)`
`-->`
terminate `store[x]`

Retrospective EBL



Concept Network

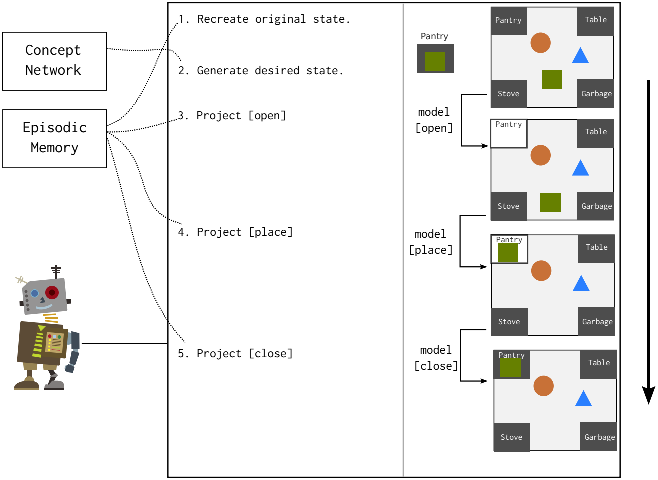
Episodic Memory

1. Recreate original state.
2. Generate desired state.
3. Project [open]
4. Project [place]

terminate store:
If store([x]) and
IN([x],PANTRY) and
CLOSED(PANTRY)
-->
terminate store[x]

Retrospective EBL

terminate store:
If store([x]) and
IN([x],PANTRY) and
CLOSED(PANTRY)
-->
terminate store[x]

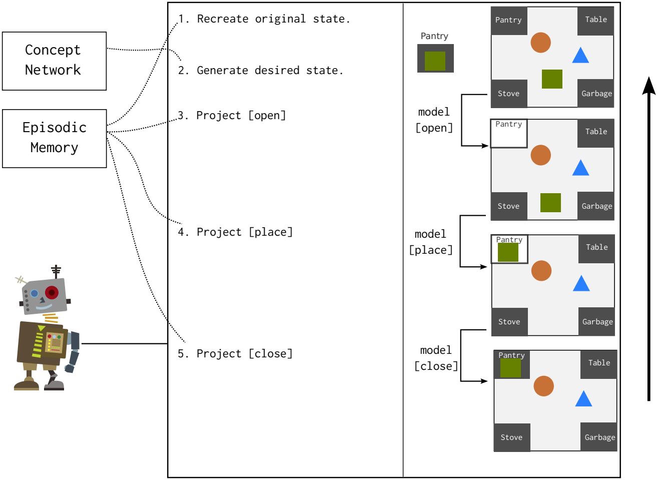


Concept Network

Episodic Memory

1. Recreate original state.
2. Generate desired state.
3. Project [open]
4. Project [place]
5. Project [close]

Retrospective EBL

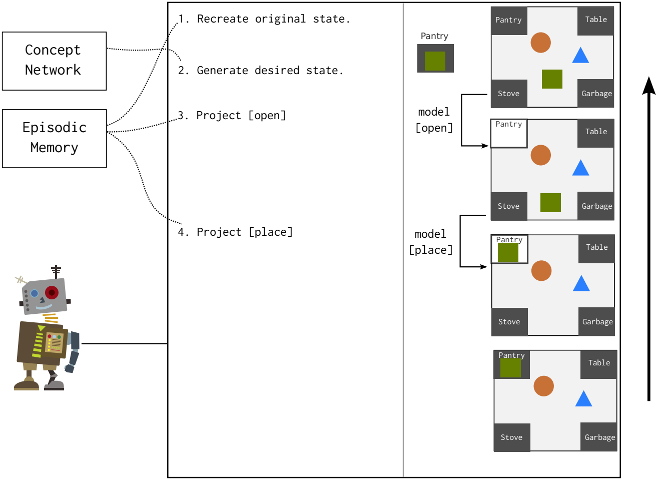


1. Recreate original state.
2. Generate desired state.
3. Project [open]
4. Project [place]
5. Project [close]

terminate store:
If `store([x])` and
`IN([x],PANTRY)` and
`CLOSED(PANTRY)`
-->
terminate `store[x]`

select close:
If `store([x])` and
`IN([x],PANTRY)` and
`OPEN(PANTRY)`
-->
select `close(PANTRY)`

Retrospective EBL



Concept Network

Episodic Memory

1. Recreate original state.
2. Generate desired state.
3. Project [open]
4. Project [place]

terminate store:
If store([x]) and
IN([x],PANTRY) and
CLOSED(PANTRY)
-->
terminate store[x]

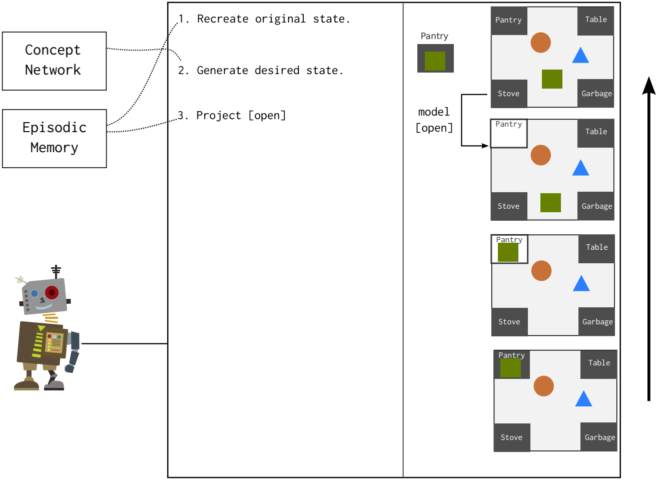
---

select close:
If store([x]) and
IN([x],PANTRY) and
OPEN(PANTRY)
-->
select close(PANTRY)

---

select place:
If store([x]) and
-IN([x],PANTRY) and
OPEN(PANTRY)
-->
select
place([x],IN,PANTRY)

Retrospective EBL



Concept Network

Episodic Memory

1. Recreate original state.

2. Generate desired state.

3. Project [open]

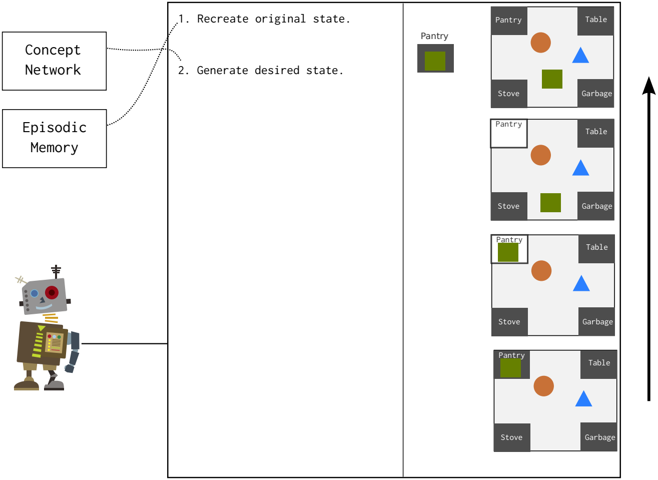
terminate store:
If store([x]) and
IN([x],PANTRY) and
CLOSED(PANTRY)
-->
terminate store[x]

select close:
If store([x]) and
IN([x],PANTRY) and
OPEN(PANTRY)
-->
select close(PANTRY)

select place:
If store([x]) and
-IN([x],PANTRY) and
OPEN(PANTRY)
-->
select
place([x],IN,PANTRY)

select open:
If store([x]) and
-IN([x],PANTRY) and
CLOSED(PANTRY)
-->
select open(PANTRY)

Retrospective EBL



Concept Network

Episodic Memory

1. Recreate original state.
2. Generate desired state.

terminate store:
If store([x]) and
IN([x],PANTRY) and
CLOSED(PANTRY)
-->
terminate store[x]

---

select close:
If store([x]) and
IN([x],PANTRY) and
OPEN(PANTRY)
-->
select close(PANTRY)

---

select place:
If store([x]) and
-IN([x],PANTRY) and
OPEN(PANTRY)
-->
select
place([x],IN,PANTRY)

---

select open:
If store([x]) and
-IN([x],PANTRY) and
CLOSED(PANTRY)
-->
select open(PANTRY)

---

available store:
If -IN([x],PANTRY) or
OPEN(PANTRY)
-->
available store([x])

Multi-task Learning

pick-up & put-down:
`place([x],[rel],[y])`, `move([x],[y])`, `discard([x])`, `store([x])`
functional:
`cook([x])`, `serve([x])`
organizational:
`stack-3([x],[y],[z])`, `stack-4([x],[y],[z],[w])`

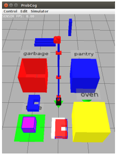
Generality in Learning

Learns general representations of tasks from few (~2-3) instances

Generality in Learning

Learns general representations of tasks from few (~2-3) instances

abstraction

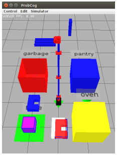


`on (purple object, table)`

Generality in Learning

Learns general representations of tasks from few (~2-3) instances

abstraction



`on (purple object, table)`

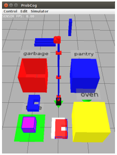
predicate selection

select open:
If `store(O1)` and `-IN(O1,PANTRY)` and
`CLOSED(PANTRY)` and ~~`CLOSED(STOVE)`~~ and
~~`OFF(STOVE)`~~ and ~~`-ON(O2,STOVE)`~~ and ...
-->
select `open(PANTRY)`

Generality in Learning

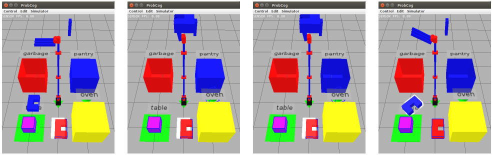
Learns general representations of tasks from few (~2-3) instances

abstraction



`on (purple object, table)`

causal analysis



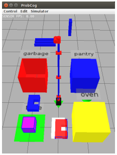
predicate selection

select open:
If `store(O1)` and `-IN(O1,PANTRY)` and
`CLOSED(PANTRY)` and ~~`CLOSED(STOVE)`~~ and
~~`OFF(STOVE)`~~ and ~~`-ON(O2,STOVE)`~~ and ...
`-->`
select `open(PANTRY)`

Generality in Learning

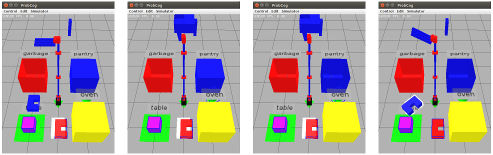
Learns general representations of tasks from few (~2-3) instances

abstraction



on (purple object, table)

causal analysis



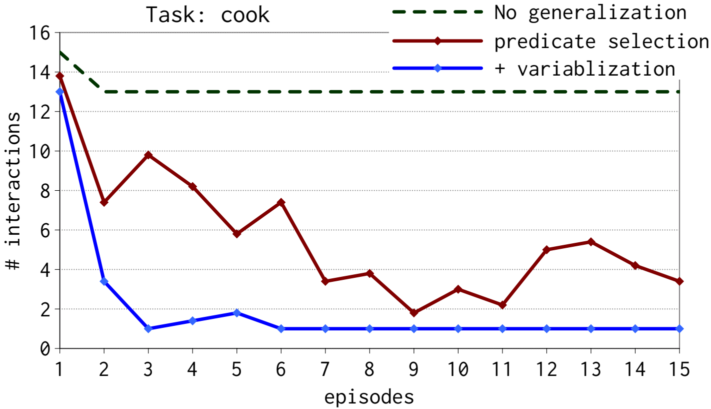
predicate selection

select open:
If store(O1) and -IN(O1,PANTRY) and CLOSED(PANTRY) and ~~CLOSED(STOVE)~~ and ~~OFF(STOVE)~~ and ~~-ON(O2,STOVE)~~ and ...
-->
select open(PANTRY)

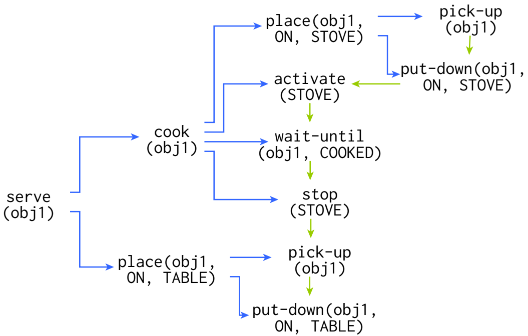
variablization

Store the green rectangle.
The goal is the green rectangle in the pantry and the pantry is closed.
Open the pantry.
Move the green rectangle to the pantry.
...

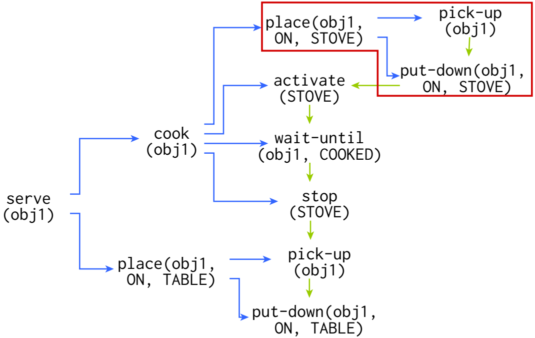
Generality in Learning



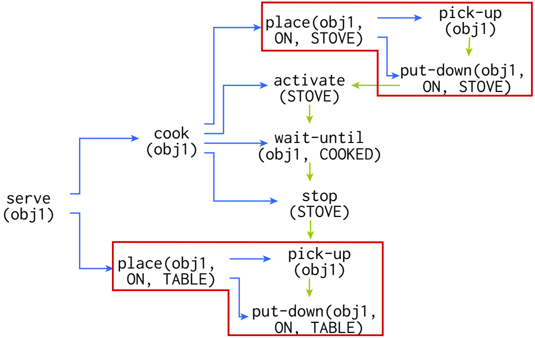
Hierarchical Learning



Hierarchical Learning



Hierarchical Learning



serve (obj1)

cook (obj1)

place(obj1, ON, STOVE)

pick-up (obj1)

put-down(obj1, ON, STOVE)

activate (STOVE)

wait-until (obj1, COOKED)

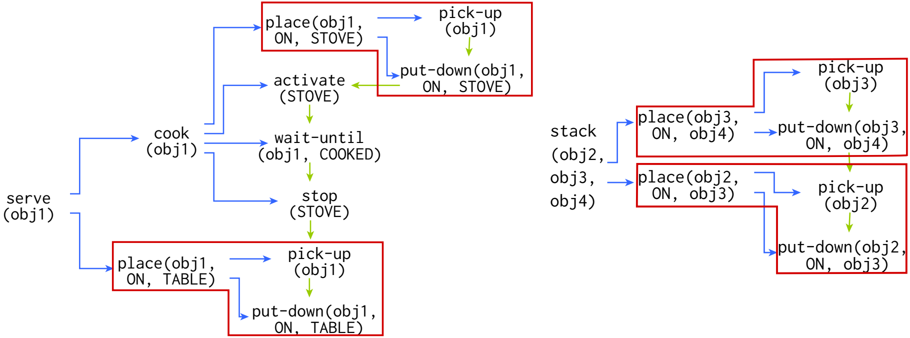
stop (STOVE)

place(obj1, ON, TABLE)

pick-up (obj1)

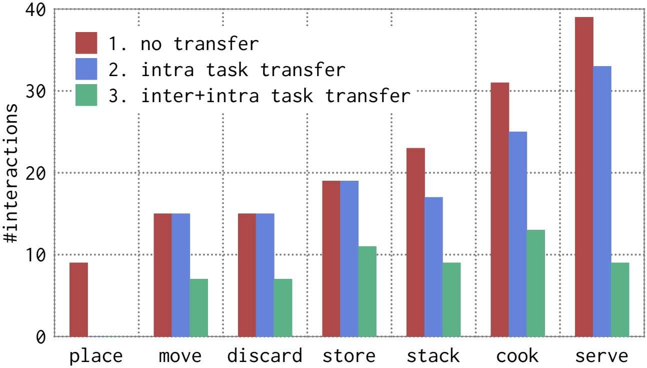
put-down(obj1, ON, TABLE)

Hierarchical Learning



Transfer

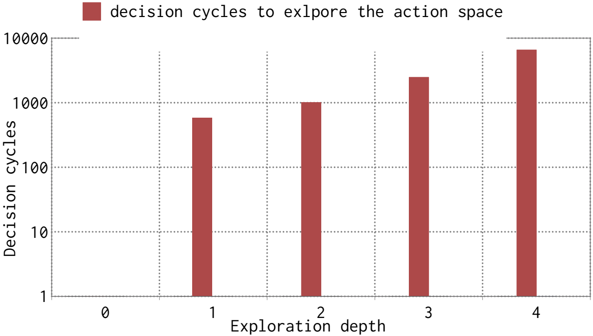
Exploits the common policy space for instruction-aided transfer.



Distributed Initiative

Integrates agent-driven exploration and instruction-guided exploitation

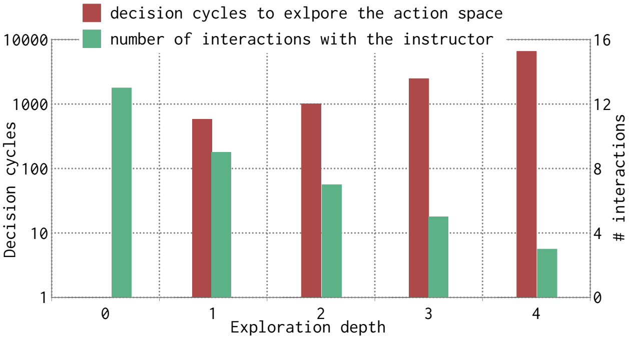
Learning the task *store*.



Distributed Initiative

Integrates agent-driven exploration and instruction-guided exploitation

Learning the task *store*.



Nuggets and Coal

Nuggets

- composable, hierarchical, transferable representation
- multi-task learning
- fast generalization
- distributed initiative of learning
- uses several Soar mechanisms
- Rosie talks!

Coal

- only *achievement* tasks
- not completely robust to instruction errors
- HRI evaluation