New Task Learning Capabilities in Rosie

Aaron Mininger

University of Michigan

May 9, 2019

2019 Soar Workshop

Interactive Task Learning

Design agents that can learn new tasks from scratch through natural forms of interaction

Situated Interactive Instruction

Situated

Instruction happens in a shared environment

Interactive

Both the instructor and agent engage in dialog

Instruction

Agent learns primarily through natural language

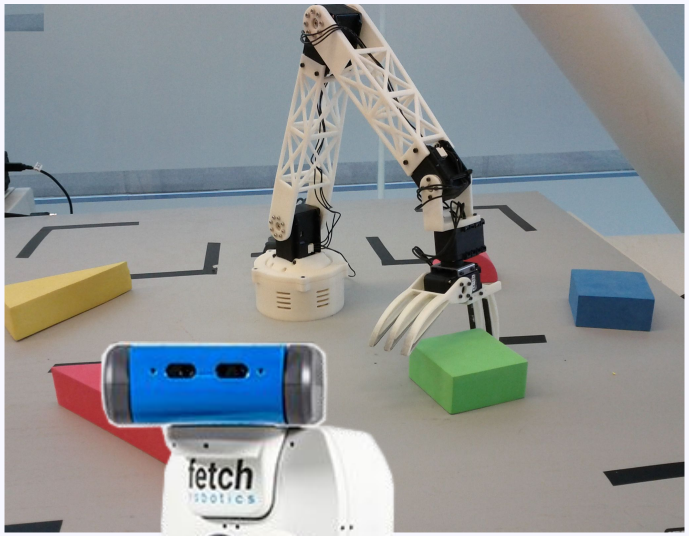
Instructional-ITL Learning Problem

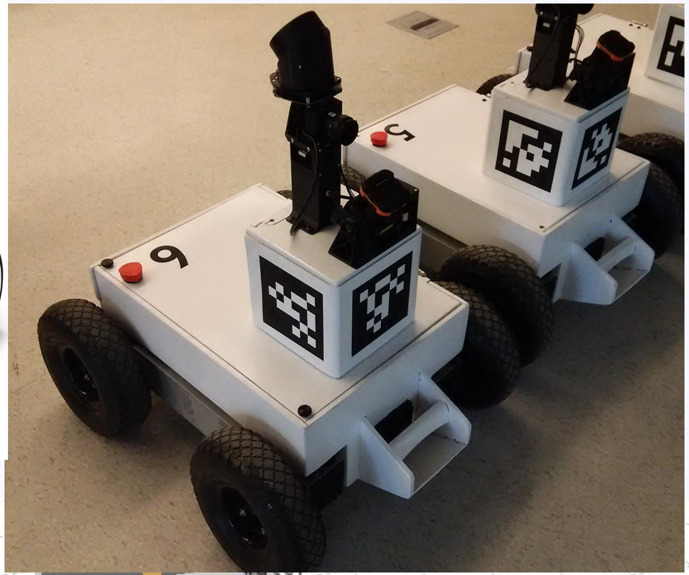
Key Characteristic: The agent must learn quickly from few examples

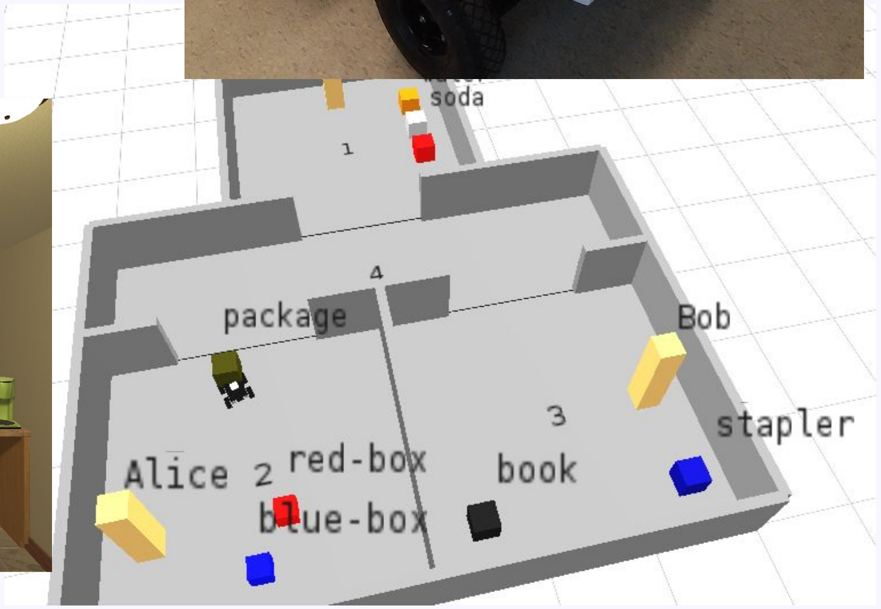
Learning must be:

- **Efficient:** Maximize learning from each instruction
- **Generalizable:** Apply learning to future task variations
- **Compositional:** Build on previously learned knowledge
- **Diverse:** Learn a range of task and knowledge types

Task Domains







Make me a cup of tea.

Schedule a meeting with Dave.

Task Diversity

Lead a tour of the office.

Deliver the mail.

Find the fastest route to a supermarket.

Task Diversity

Tasks will have a variety of different

- Characteristics
- Objects
- Concepts
- Actions
- Instructional Strategies

Key Research Question

How to expand the diversity of tasks an Instructional-ITL agent can learn?

Three main dimensions of complexity:

- **Diverse Action Types**
- **Diverse Task Modifiers**
- **Diverse Task Formulations**

Demo

Key Research Question

How to expand the diversity of tasks an Instructional-ITL agent can learn?

Three main dimensions of complexity:

- **Diverse Action Types**
- **Diverse Task Modifiers**
- **Diverse Task Formulations**

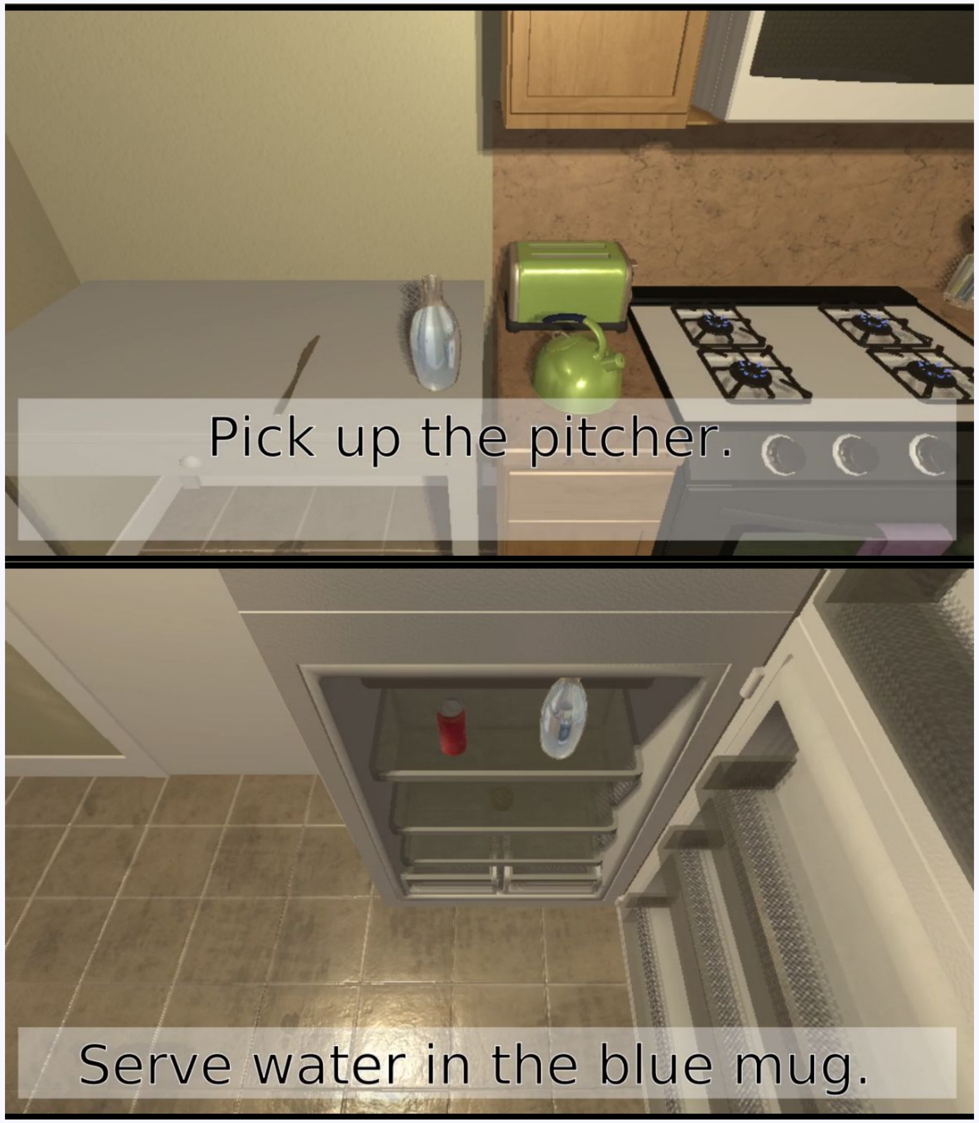
Diverse Action Types

Want to teach tasks with a variety of types of actions:

- **Physical**
- **Perceptual**
- **Communicative**
- **Memory-Based**

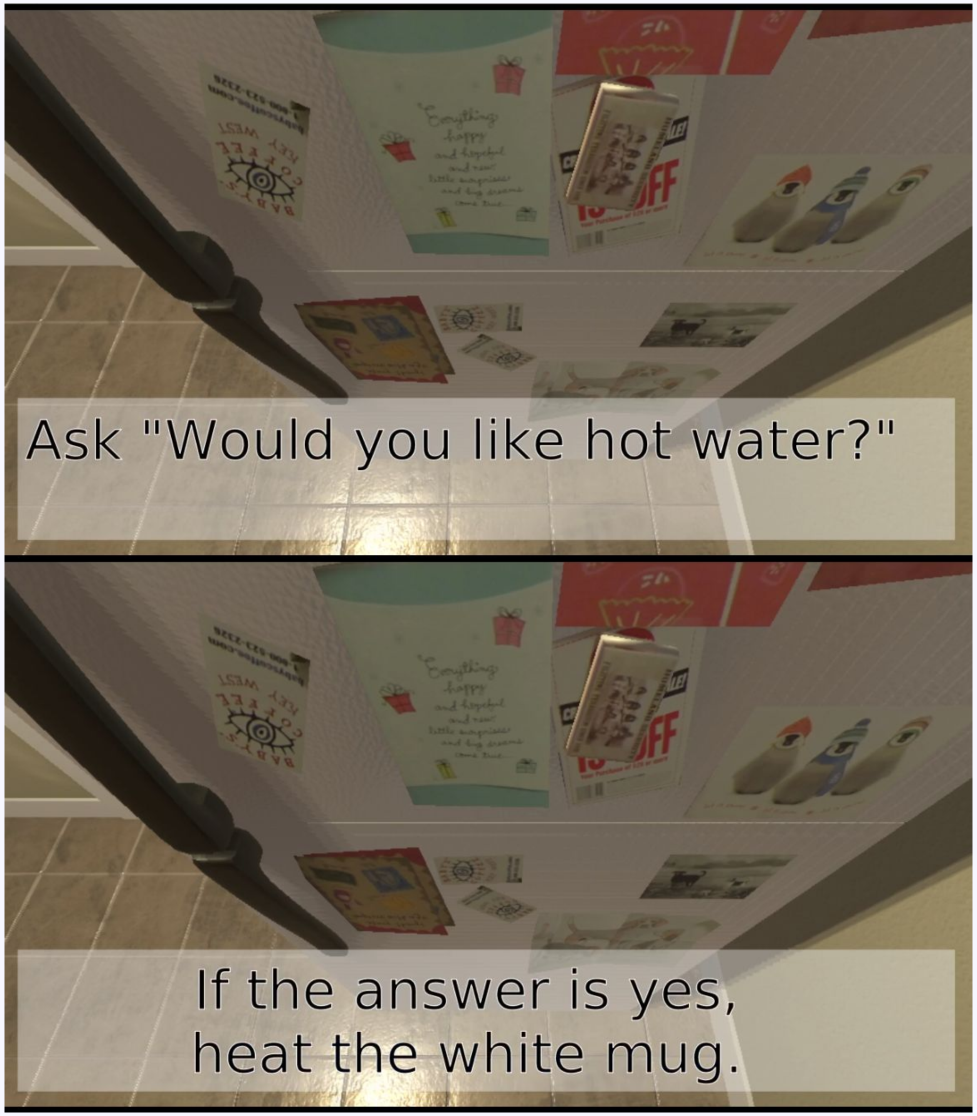
Diverse Action Types

Perceptual



Diverse Action Types

Communicative


Ask "Would you like hot water?"
If the answer is yes, heat the white mug.

Diverse Task Formulations

How is the task represented?
What learning mechanisms are used?

Diverse Task Formulations

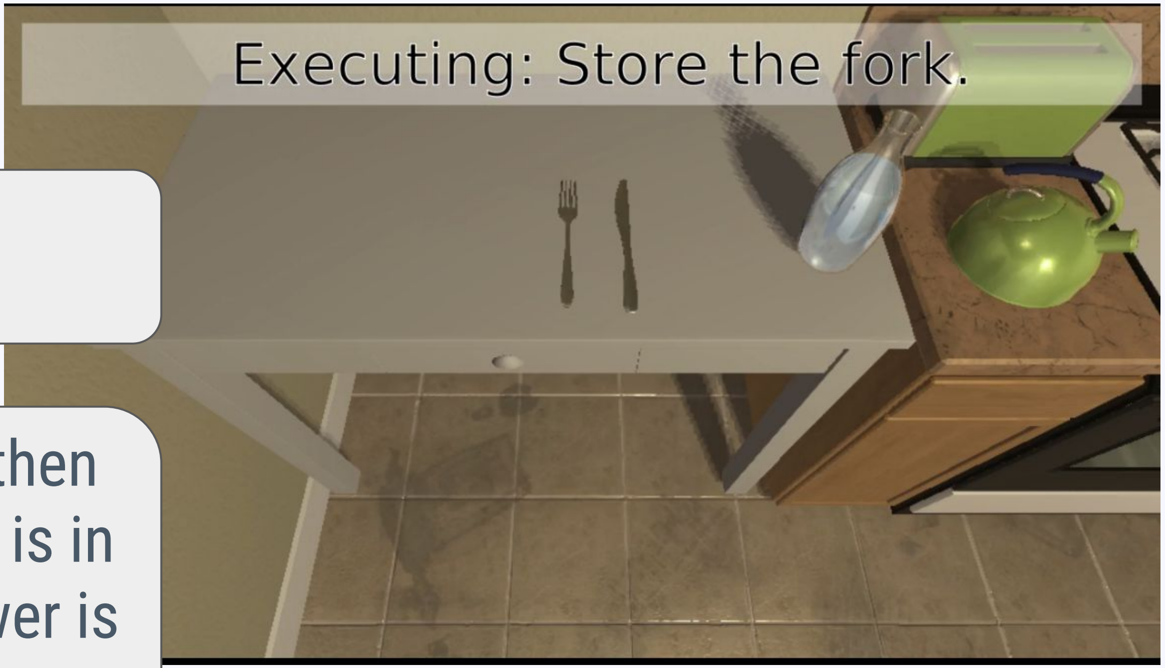
How is the task represented?
What learning mechanisms are used?

May depend on:

- How the task is taught
- What capabilities the agent has
- Characteristics of the task

Diverse Task Formulations

Goal-Based



Store the fork.

If the fork is a utensil, then the goal is that the fork is in the drawer and the drawer is closed.

Diverse Task Formulations

Procedural

Heat the blue mug in the microwave.

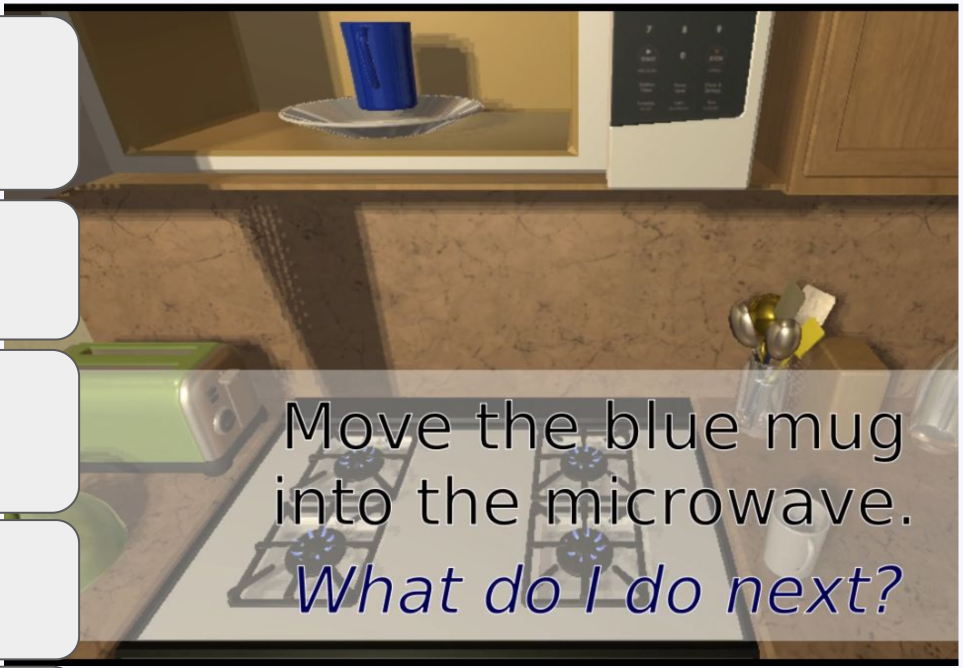
Open the microwave.

Move the blue mug into the microwave.

Close the microwave.

Turn on the microwave for five seconds.

Wait until the microwave is off.



Diverse Task Modifiers

There are many ways a task can be modified:

- **Temporal:** *when* a task should be performed
- **Conditional:** *whether* a task should be performed
- **Repetitious:** *how many times* a task should be performed
- **Spatial:** *where* a task should be performed
- **Manner:** *how* a task should be performed

Temporal Modifiers

- **Start**

  At 3:00, open the door.
  After three minutes, turn off the lights.
  After the microwave is off, open the door.

- **Duration**

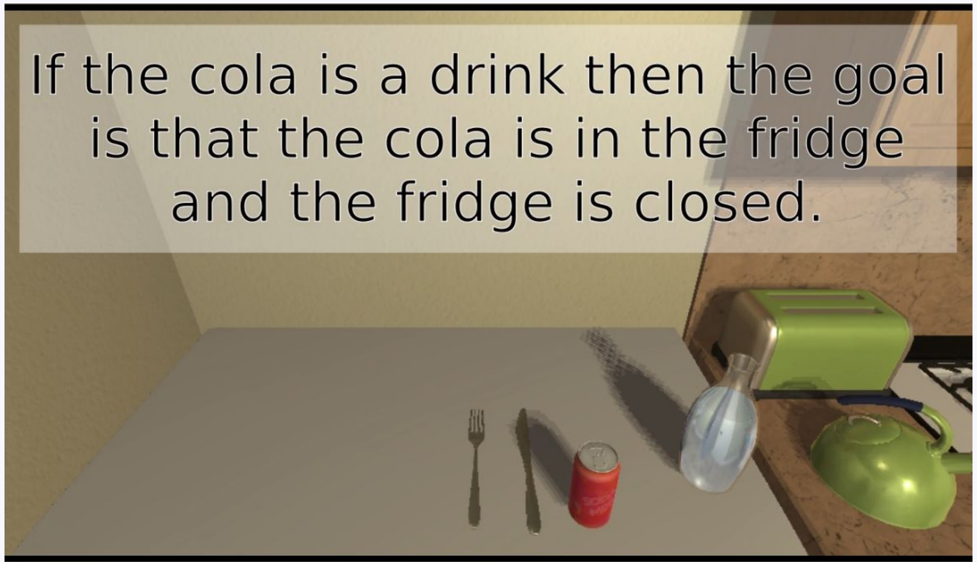
  Wait for one minute.
  Turn on the microwave for five seconds.

- **End**

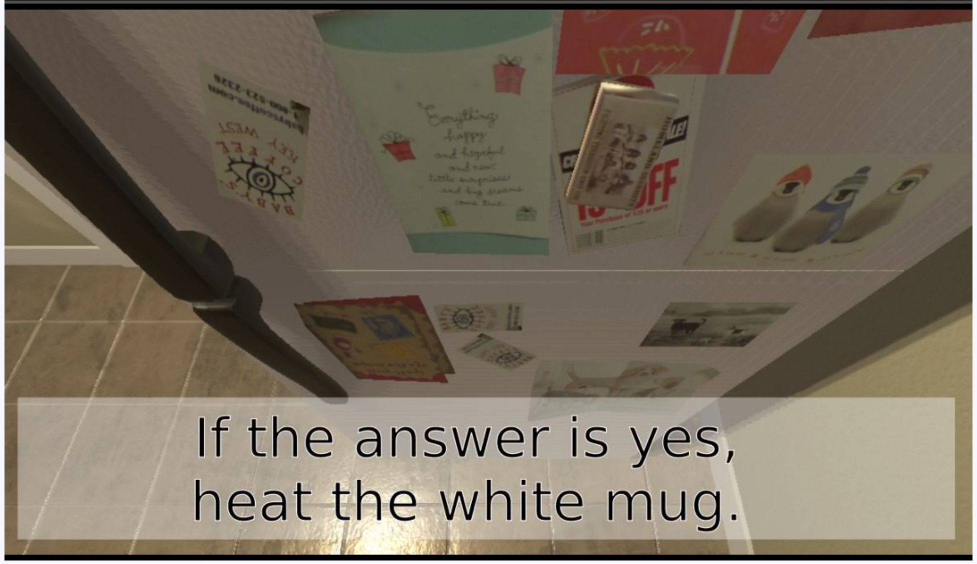
  Wait until 4:00.
  Turn right until you see the door.
  Press the down button until the screen is lowered.

Conditional Modifiers

- **Conditional Goals**

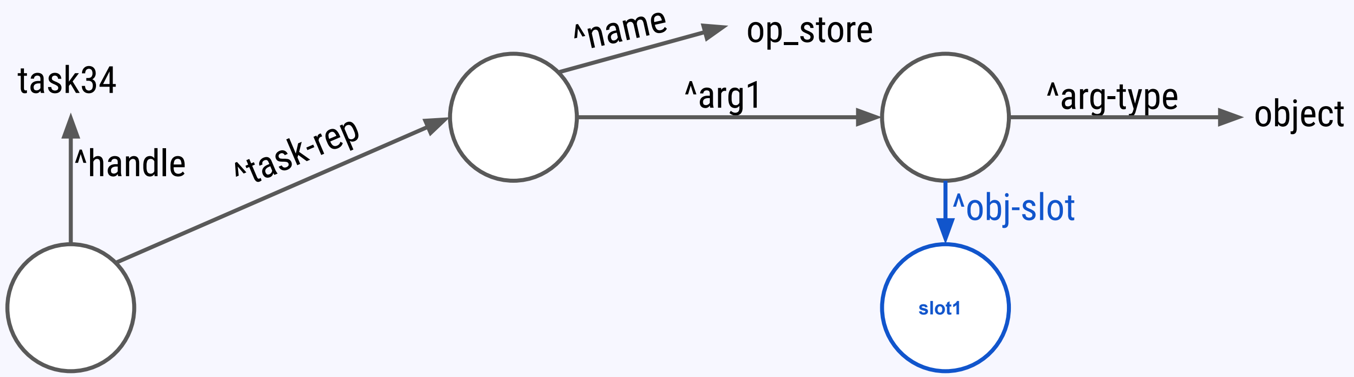

If the cola is a drink then the goal is that the cola is in the fridge and the fridge is closed.

- **Conditional Actions**



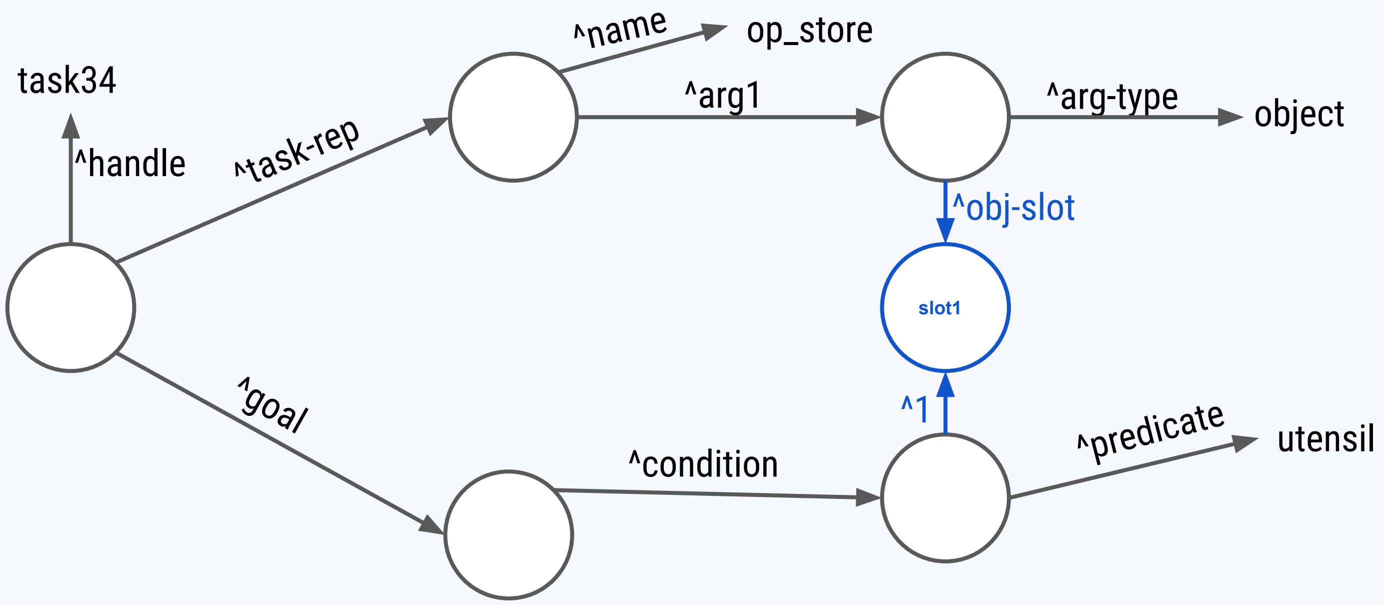
Learning Conditional Goals

Store the fork.



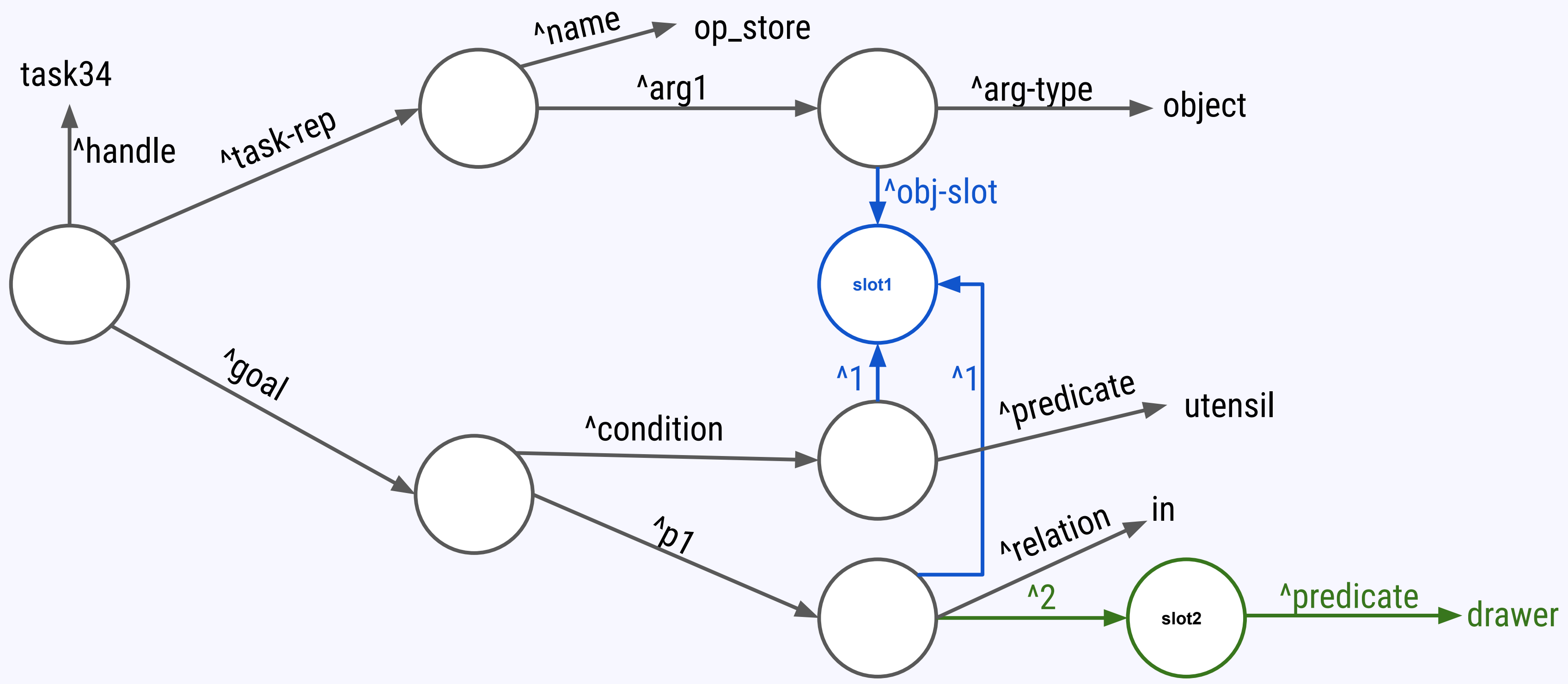
Learning Conditional Goals

If the fork is a utensil



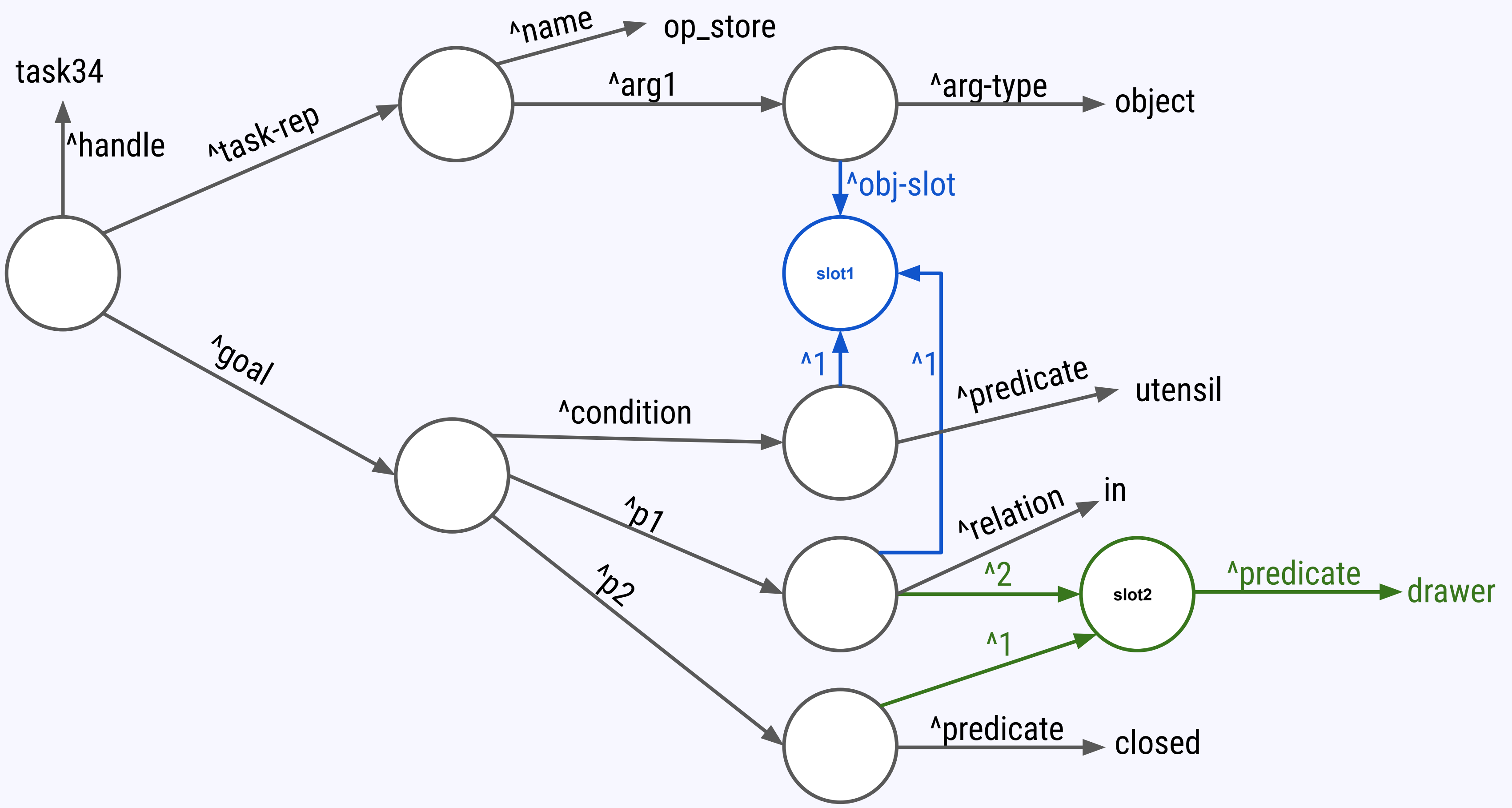
Learning Conditional Goals

If the fork is a utensil, then the goal is that the fork is in the drawer



Learning Conditional Goals

If the fork is a utensil, then the goal is that the fork is in the drawer and the drawer is closed.



Learning Conditional Goals

If the fork is a utensil, then the goal is that the fork is in the drawer and the drawer is closed.

```
(<tcn> ^handle store ^task-rep <task-rep> ^goal <goal>)

  (<task-rep> ^name op_store ^arg1 <arg1>)
     (<arg1> ^arg-type object ^id <slot1>)

  (<goal> ^conditions <conds> ^p1 <p1> ^p2 <p2>)
     (<conds> ^p1 <cond-p1>)
        (<cond-p1> ^predicate utensil ^1 <slot1>)

     (<p1> ^relation in ^1 <slot1> ^2 <slot2>)
     (<p2> ^predicate closed ^2 <slot2>)
        (<slot2> ^predicate drawer)
```

Learning Conditional Goals

If the fork is a utensil, then the goal is that the fork is in the drawer and the drawer is closed.

```
(state <s> ^name op_store
           ^current-task <task>
           ^world <world>
           ^task-concept-network <tcn>)

   (<task> ^name op_store
           ^arg1 <arg1>)
       (<arg1> ^arg-type object ^id <obj1>)

   (<world> ^objects <objs>)
      (<objs> ^object <obj1> <obj2> <obj3> ... )
         (<obj1> ^handle obj25 ^predicate fork gray utensil visible ...)
         (<obj2> ^handle obj48 ^predicate drawer closed not-visible ... )
```

Learning Conditional Goals

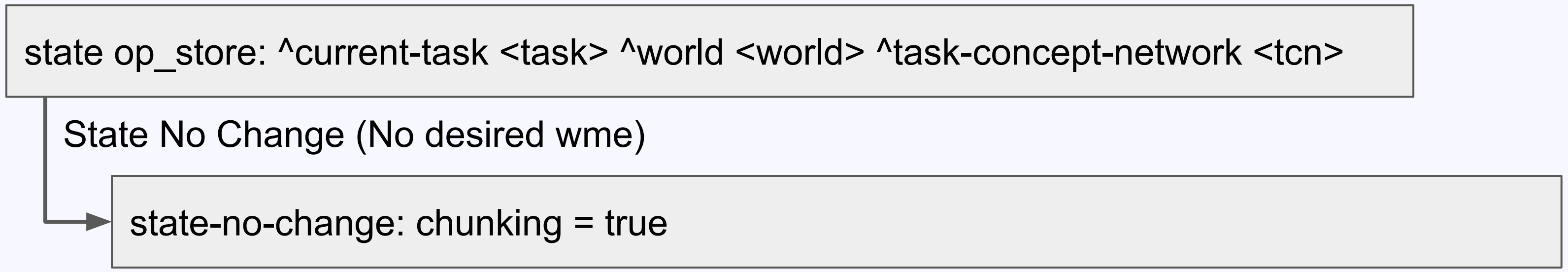
If the fork is a utensil, then the goal is that the fork is in the drawer and the drawer is closed.

```
(<task> = op_store(arg1=<obj1>) )
(<world> = { <obj1> <obj2> ... })
(<obj1> = { fork, gray, utensil, visible ... })
(<obj2> = { drawer, closed, not-visible, ... })
```

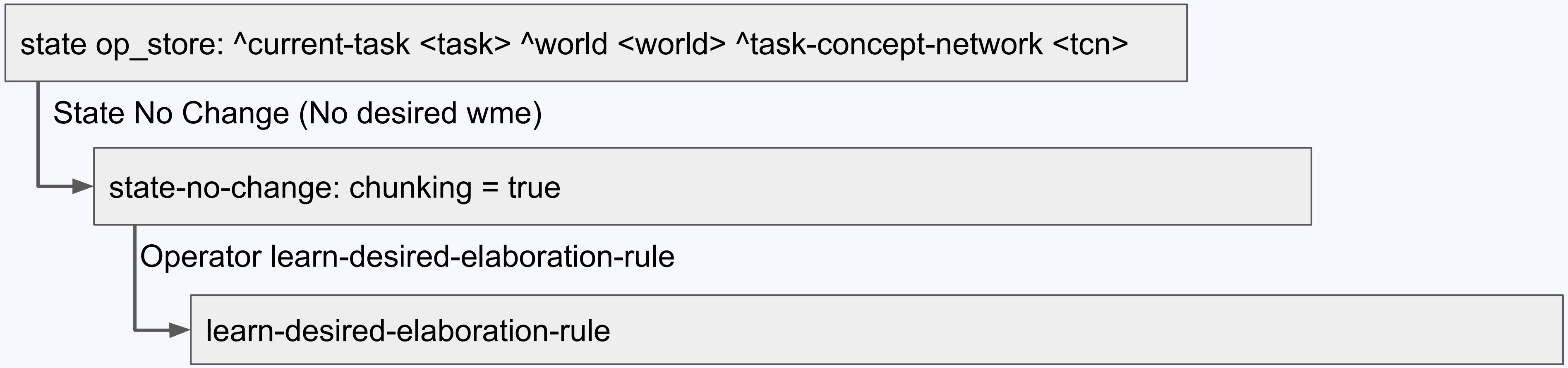
Learning Conditional Goals

```
state op_store: ^current-task <task> ^world <world> ^task-concept-network <tcn>
```

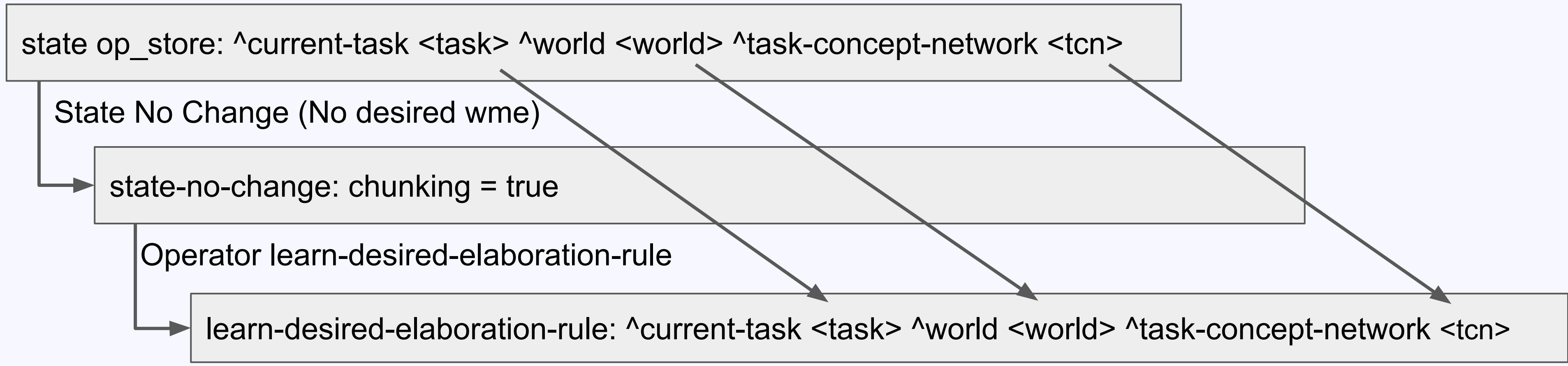
Learning Conditional Goals


state op_store: ^current-task <task> ^world <world> ^task-concept-network <tcn>
State No Change (No desired wme)
state-no-change: chunking = true

Learning Conditional Goals


state op_store: ^current-task <task> ^world <world> ^task-concept-network <tcn>
State No Change (No desired wme)
state-no-change: chunking = true
Operator learn-desired-elaboration-rule
learn-desired-elaboration-rule

Learning Conditional Goals


state op_store: ^current-task <task> ^world <world> ^task-concept-network <tcn>
State No Change (No desired wme)
state-no-change: chunking = true
Operator learn-desired-elaboration-rule
learn-desired-elaboration-rule: ^current-task <task> ^world <world> ^task-concept-network <tcn>

Learning Conditional Goals

```
learn-desired-elaboration-rule: ^current-task <task> ^world <world> ^task-concept-network <tcn>
```

```
Result: (to superstate)
```

```
<task> = op_store(arg1=<obj1>)
<world> = { <obj1> <obj2> }
<obj1> = { fork, gray, utensil, visible }
<obj2> = { drawer, closed, not-visible }
```

```
(<tcn> ^task-rep <task-rep> ^goal <goal>)
(<task-rep> ^name op_store ^arg1 <arg1>)
   (<arg1> ^arg-type object ^id <slot1>)
(<goal> ^conditions <conds> ^p1 <p1> ^p2 <p2>)
   (<conds> ^p1 <cond-p1>)
      (<cond-p1> ^predicate utensil ^1 <slot1>)
   (<p1> ^relation in ^1 <slot1> ^2 <slot2>)
   (<p2> ^predicate closed ^2 <slot2>)
      (<slot2> ^predicate drawer)
```

Learning Conditional Goals

learn-desired-elaboration-rule: ^current-task <task> ^world <world> ^task-concept-network <tcn>

```
Result: (to superstate)
(<s> ^desired <des>)
(<des> ^p1 <p1> ^p2 <p2>)
(<p1> ^relation in                    )
(<p2> ^property closed             )
```

```
<task> = op_store(arg1=<obj1>)
<world> = { <obj1> <obj2> }
<obj1> = { fork, gray, utensil, visible }
<obj2> = { drawer, closed, not-visible }
```

```
(<tcn> ^task-rep <task-rep> ^goal <goal>)
(<task-rep> ^name op_store ^arg1 <arg1>)
   (<arg1> ^arg-type object ^id <slot1>)
(<goal> ^conditions <conds> ^p1 <p1> ^p2 <p2>)
   (<conds> ^p1 <cond-p1>)
      (<cond-p1> ^predicate utensil ^1 <slot1>)
   (<p1> ^relation in ^1 <slot1> ^2 <slot2>)
   (<p2> ^predicate closed ^2 <slot2>)
      (<slot2> ^predicate drawer)
```

Learning Conditional Goals

```
learn-desired-elaboration-rule: ^current-task <task> ^world <world> ^task-concept-network <tcn>
```

```
Result: (to superstate)
(<s> ^desired <des>)
(<des> ^p1 <p1> ^p2 <p2>)
(<p1> ^relation in ^1 <obj1>          )
(<p2> ^property closed        )
```

```
<task> = op_store(arg1=<obj1>)
<world> = { <obj1> <obj2> }
<obj1> = { fork, gray, utensil, visible }
<obj2> = { drawer, closed, not-visible }
```

```
(<tcn> ^task-rep <task-rep> ^goal <goal>)
(<task-rep> ^name op_store ^arg1 <arg1>)
   (<arg1> ^arg-type object ^id <slot1>)
(<goal> ^conditions <conds> ^p1 <p1> ^p2 <p2>)
   (<conds> ^p1 <cond-p1>)
      (<cond-p1> ^predicate utensil ^1 <slot1>)
   (<p1> ^relation in ^1 <slot1> ^2 <slot2>)
   (<p2> ^predicate closed ^2 <slot2>)
      (<slot2> ^predicate drawer)
```

Learning Conditional Goals

learn-desired-elaboration-rule: ^current-task <task> ^world <world> ^task-concept-network <tcn>

```
Result: (to superstate)
(<s> ^desired <des>)
(<des> ^p1 <p1> ^p2 <p2>)
(<p1> ^relation in ^1 <obj1> ^2 <obj2>)
(<p2> ^property closed ^1 <obj2>)
```

```
<task> = op_store(arg1=<obj1>)
<world> = { <obj1> <obj2> }
<obj1> = { fork, gray, utensil, visible }
<obj2> = { drawer, closed, not-visible }
```

```
(<tcn> ^task-rep <task-rep> ^goal <goal>)
(<task-rep> ^name op_store ^arg1 <arg1>)
   (<arg1> ^arg-type object ^id <slot1>)
(<goal> ^conditions <conds> ^p1 <p1> ^p2 <p2>)
   (<conds> ^p1 <cond-p1>)
      (<cond-p1> ^predicate utensil ^1 <slot1>)
   (<p1> ^relation in ^1 <slot1> ^2 <slot2>)
   (<p2> ^predicate closed ^2 <slot2>)
      (<slot2> ^predicate drawer)
```

Learning Conditional Goals

learn-desired-elaboration-rule: ^current-task <task> ^world <world> ^task-concept-network <tcn>

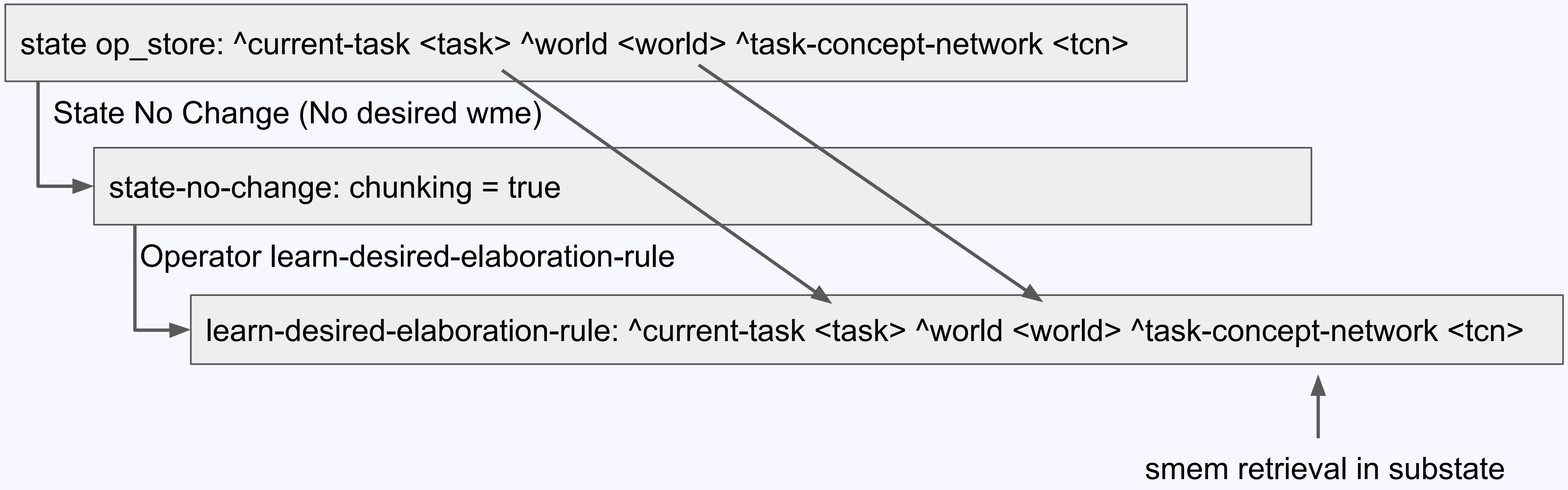
```
sp {CHUNK*op_store*elaborate*goal
   (state <s> ^name op_store
              ^current-task <task>
              ^world <world>)
   (<task> ^name op_store
           ^arg1.id <obj_A>)
   (<obj_A> ^predicate utensil)
   (<world> ^objects.object <obj_B>)
   (<obj_B> ^predicate drawer)
-->
   (<s> ^desired <des>)
   (<des> ^p1 <p1> ^p2 <p2>)
   (<p1> ^relation in ^1 <obj_A> ^2 <obj_B>)
   (<p2> ^predicate closed ^1 <obj_B>)
}
```

Learning Conditional Goals

learn-desired-elaboration-rule: ^current-task <task> ^world <world> ^task-concept-network <tcn>

```
sp {CHUNK*op_store*elaborate*goal
   (state <s> ^name op_store
              ^current-task <task>
              ^world <world>
              ^task-concept-network <tcn>)
   (<task> ^name op_store
           ^arg1.id <obj_A>)
   (<obj_A> ^predicate utensil)
   (<world> ^objects.object <obj_B>)
   (<obj_B> ^predicate drawer)
   (<tcn> ^task-rep <task-rep> ^goal <goal>)
      (<task-rep> ^name op_store ^arg1 <arg1>)
         (<arg1> ^arg-type object ^id <slot1>)
      (<goal> ^conditions <conds> ^p1 <p1> ^p2 <p2>)
         (<conds> ^p1 <cond-p1>)
            (<cond-p1> ^predicate utensil ^1 <slot1>)
         (<p1> ^relation in ^1 <slot1> ^2 <slot2>)
         (<p2> ^property closed ^2 <slot2>)
            (<slot2> ^predicate drawer)
-->
   (<s> ^desired <des>)
   (<des> ^p1 <p1> ^p2 <p2>)
   (<p1> ^relation in ^1 <obj_A> ^2 <obj_B>)
   (<p2> ^predicate closed ^1 <obj_B>)
}
```

Learning Conditional Goals


state op_store: ^current-task <task> ^world <world> ^task-concept-network <tcn>
State No Change (No desired wme)
state-no-change: chunking = true
Operator learn-desired-elaboration-rule
learn-desired-elaboration-rule: ^current-task <task> ^world <world> ^task-concept-network <tcn>
smem retrieval in substate

Learning Conditional Goals

```
sp {CHUNK*op_store*elaborate*goal
   (state <s> ^name op_store
              ^current-task <task>
              ^world <world>)
   (<task> ^name op_store
           ^arg1.id <obj_A>)
   (<obj_A> ^predicate utensil)
   (<world> ^objects.object <obj_B>)
   (<obj_B> ^predicate drawer)
-->
   (<s> ^desired <des>)
   (<des> ^p1 <p1> ^p2 <p2>)
   (<p1> ^relation in ^1 <obj_A> ^2 <obj_B>)
   (<p2> ^predicate closed ^1 <obj_B>)
}
```

Nuggets and Coal

Nuggets

- Able to learn more diverse tasks
- Integrated approach allows interesting task composition
- Successfully applied to a number of domains

Coal

- Task complexity and diversity dwarfs even these extensions
- Increased complexity leads to more unintended side effects
- Instruction uses precise wording and requires an expert

Questions?