Learning by Observation Using Inductive Logic Programming: Recent Progress

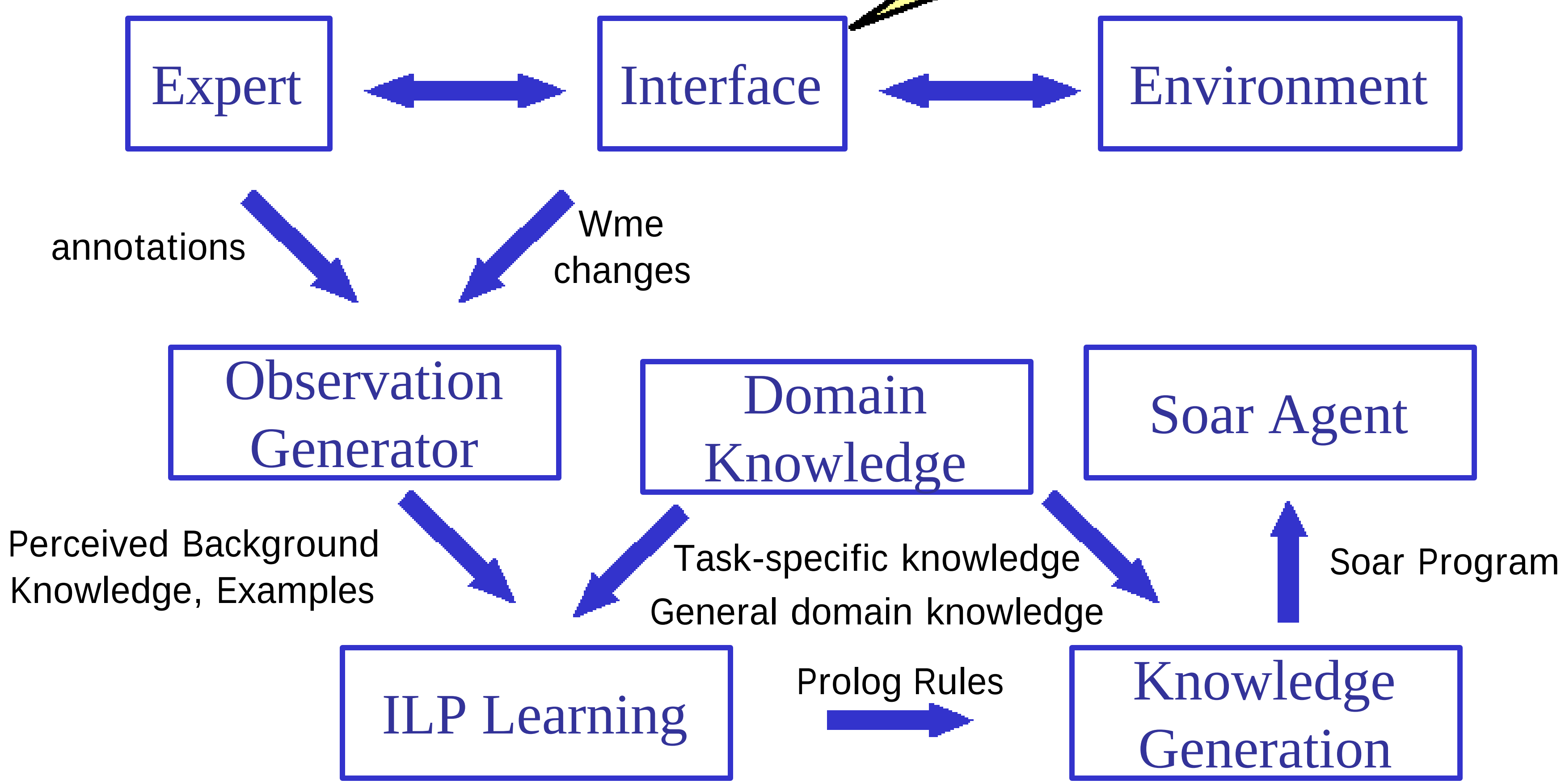
Tolga Könik
konik@umich.edu
University of Michigan

What We Want to Learn?

- We want to learn when to select operators, and how to execute them after passively observing an expert.

Learning By O

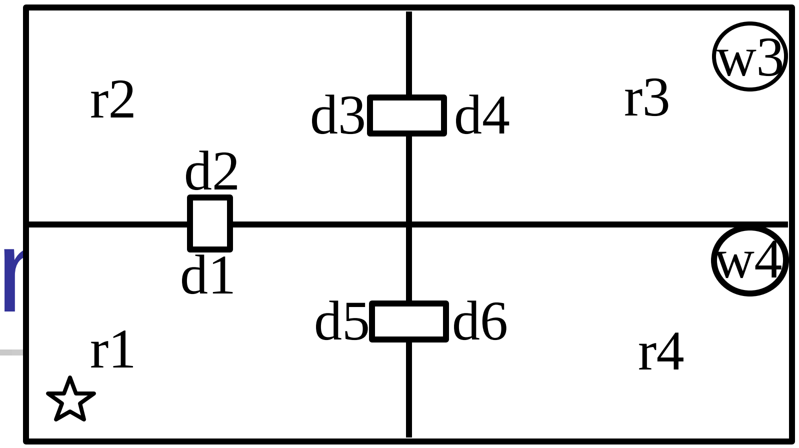
THIS CAN INCLUDE A PARTIAL SOAR PROGRAM SUCH AS A MAP BUILDER



Why Inductive Logic Programming?

- Structured domain knowledge
- Structured sensors
- Partially hand-coded agents

Expert Ann



- Expert should indicate which part of the hierarchy he/she is executing

  Get-item(w3)

  Goto-door(d1)

  ...

- The interface should have a representation for the objects that the expert might use in operator parameters

Domain Knowledge

- Common Sense Knowledge

  i.e. shortest-path between rooms

- Specific Task Knowledge

  i.e. which rooms are connected

- ~~Planning Knowledge~~

  ~~if you go through a door d1 in room r1, current-room sensor points to room r2~~

Background Knowledge in Learning

- Domain knowledge from interface

  i.e. structured sensor which gives the path between each room

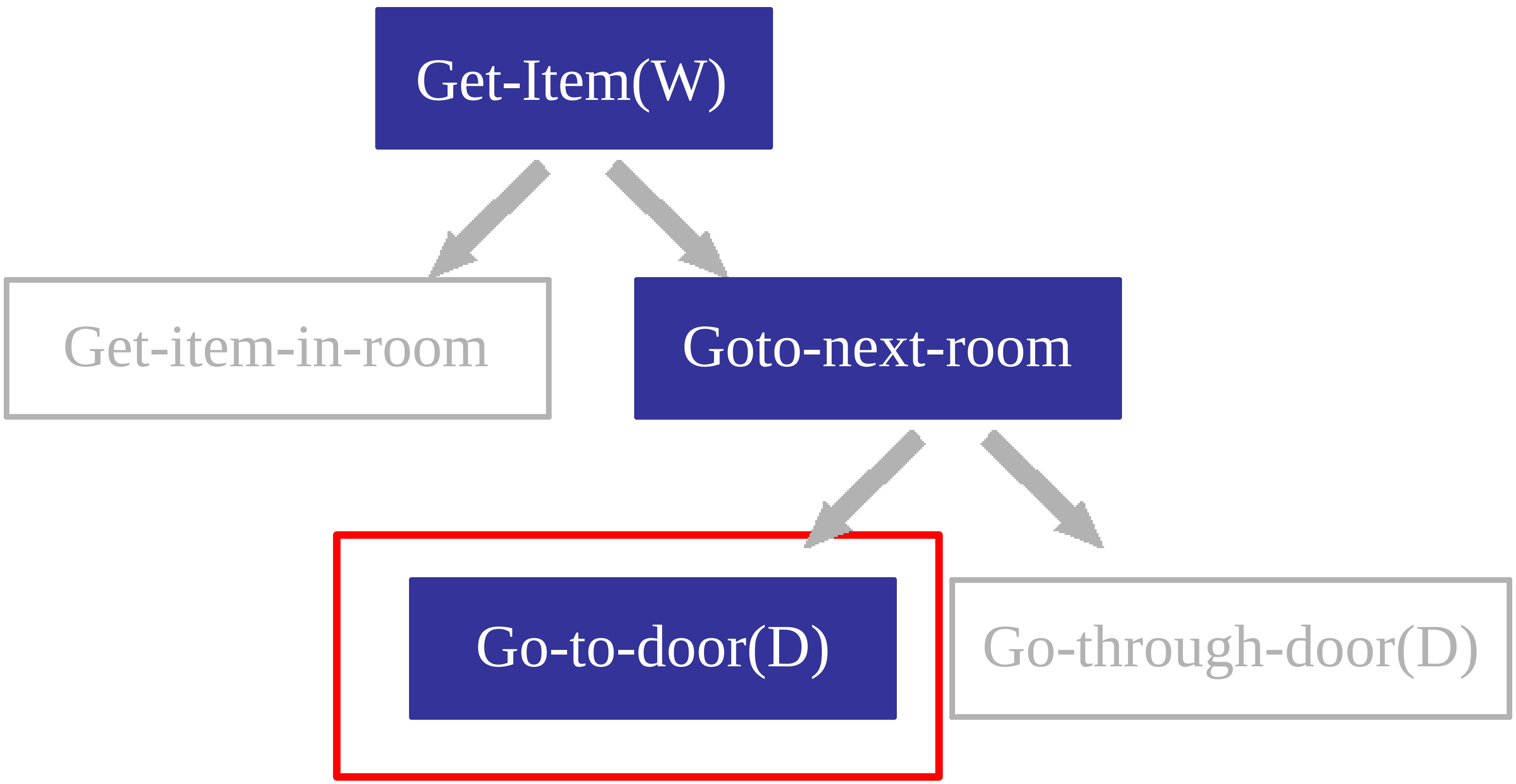
Domain knowledge can be entered to the database of the learning system

A logic program, that calculates the shortest path between two rooms for any given map

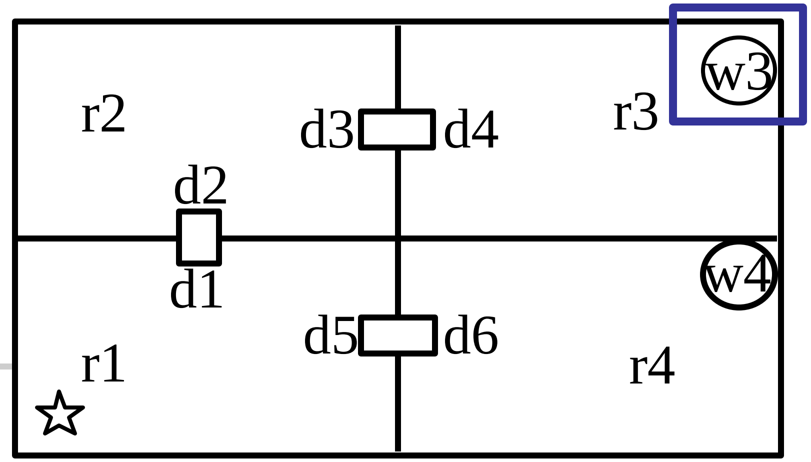
Background Knowledge in Learning

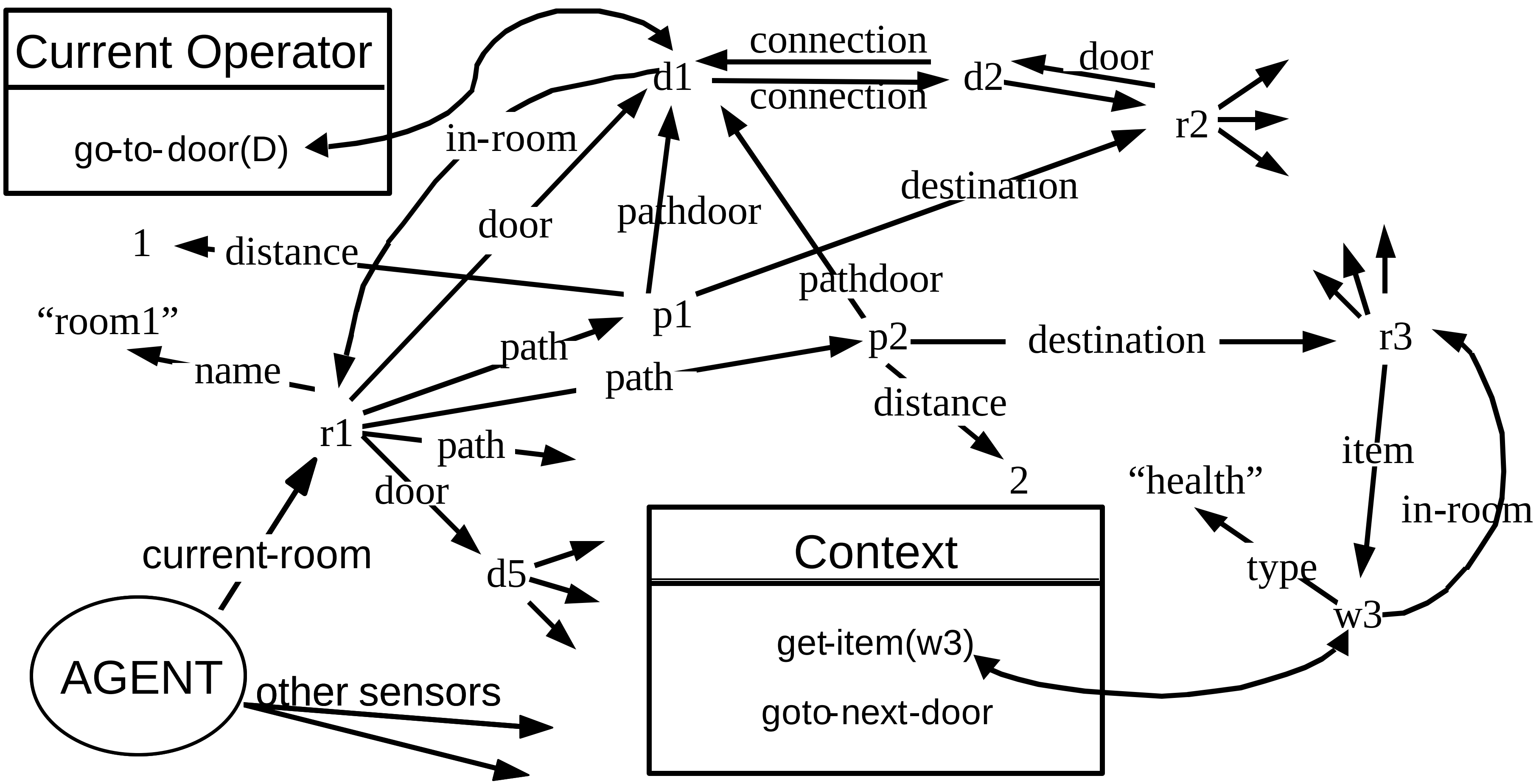
- IF domain knowledge is not part of the interface and entered to the database:
  - There should be a corresponding Soar implementation

Using Parameters of High Level Operators

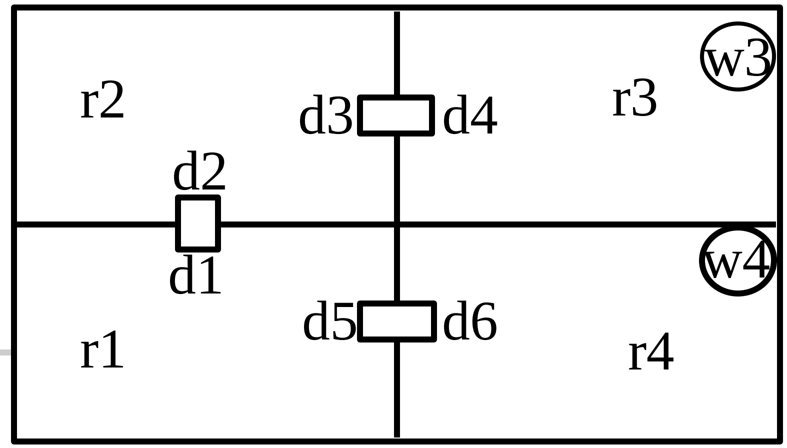


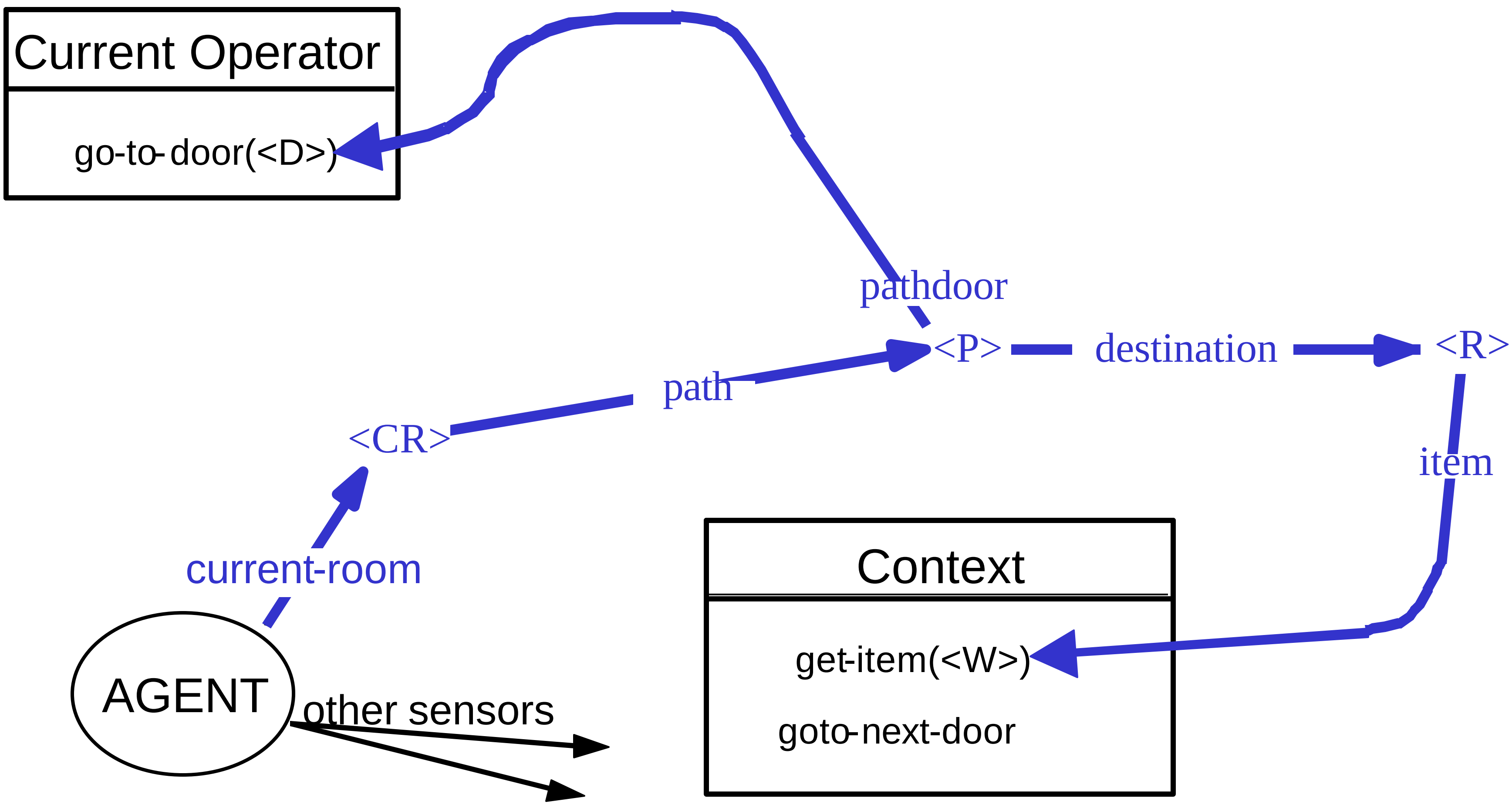
Precondition of Go-to-Door?



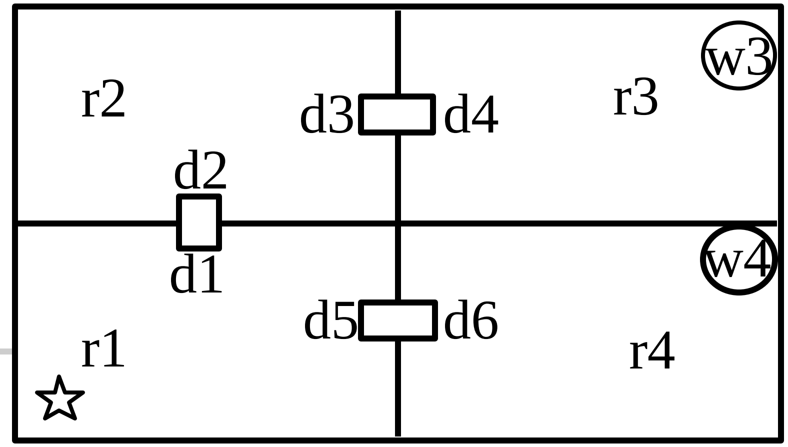


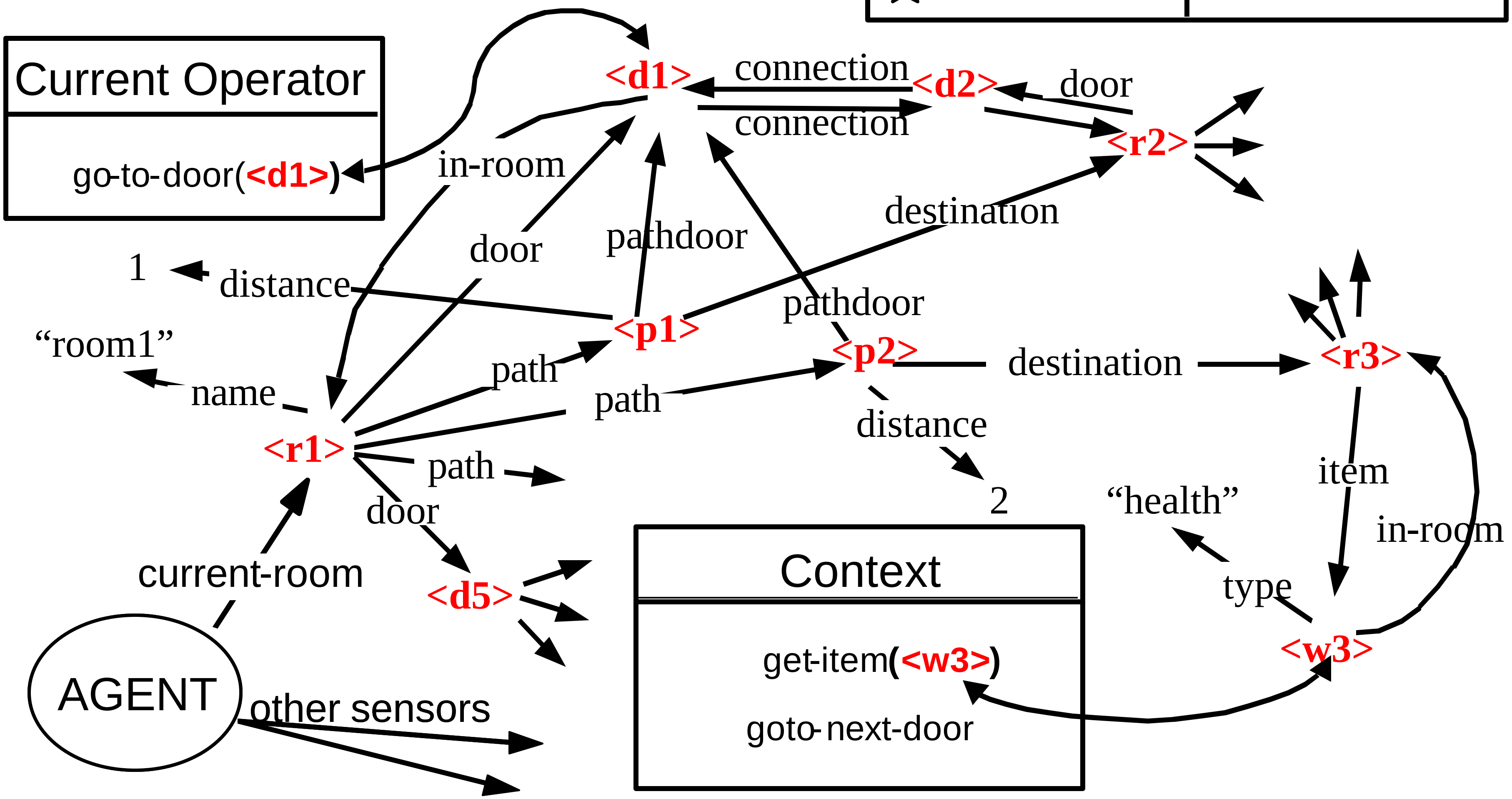
Generalization



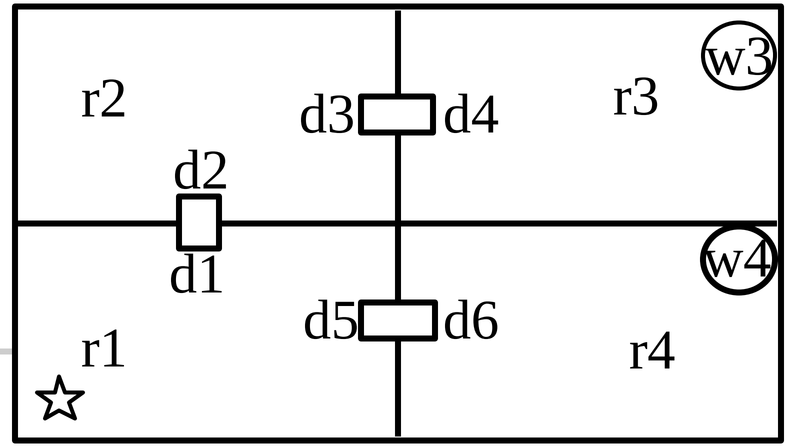


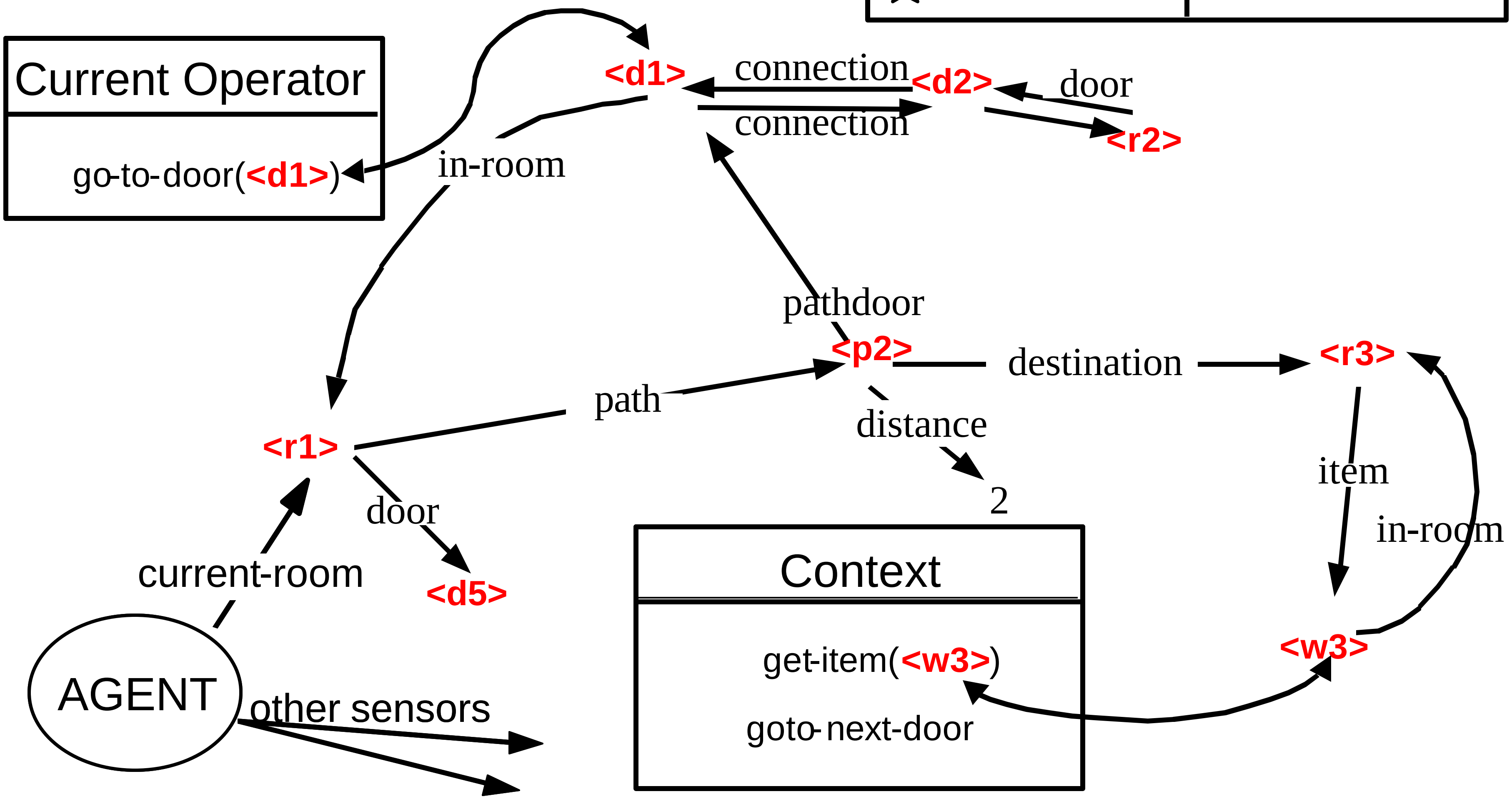
Most Specific Precondition



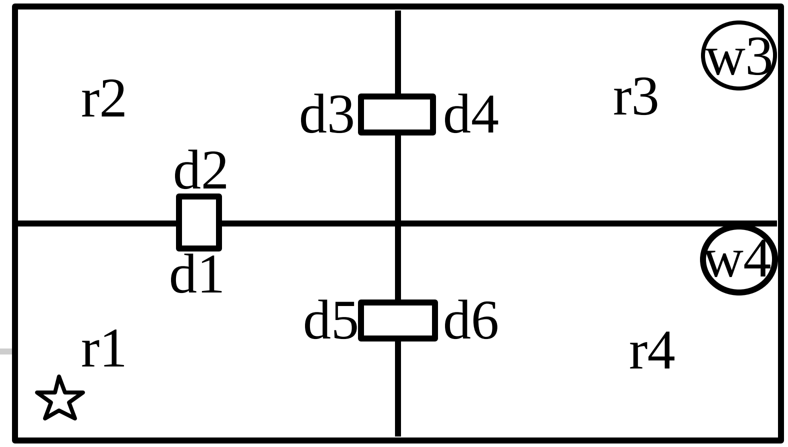


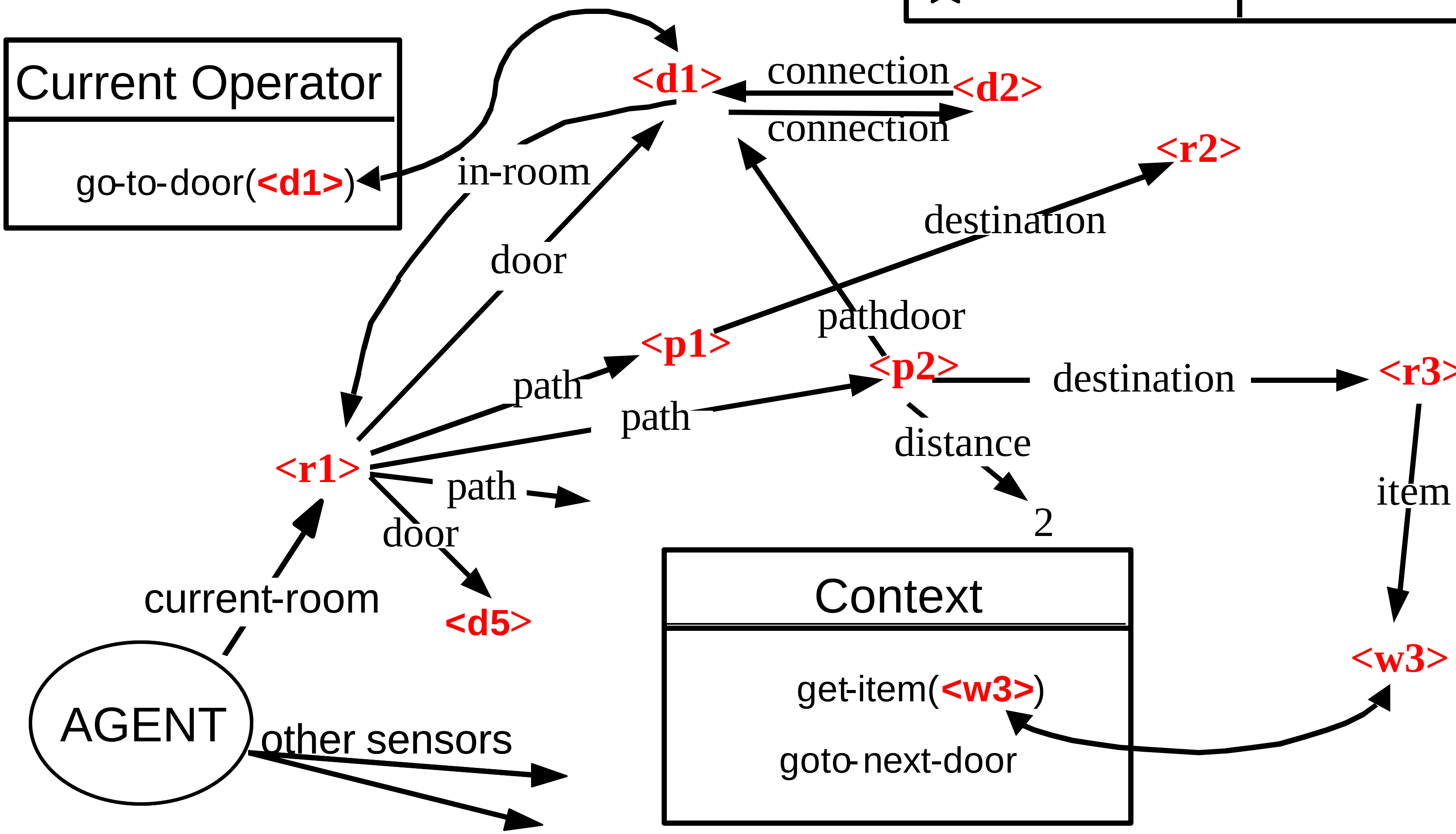
An Example Refinement



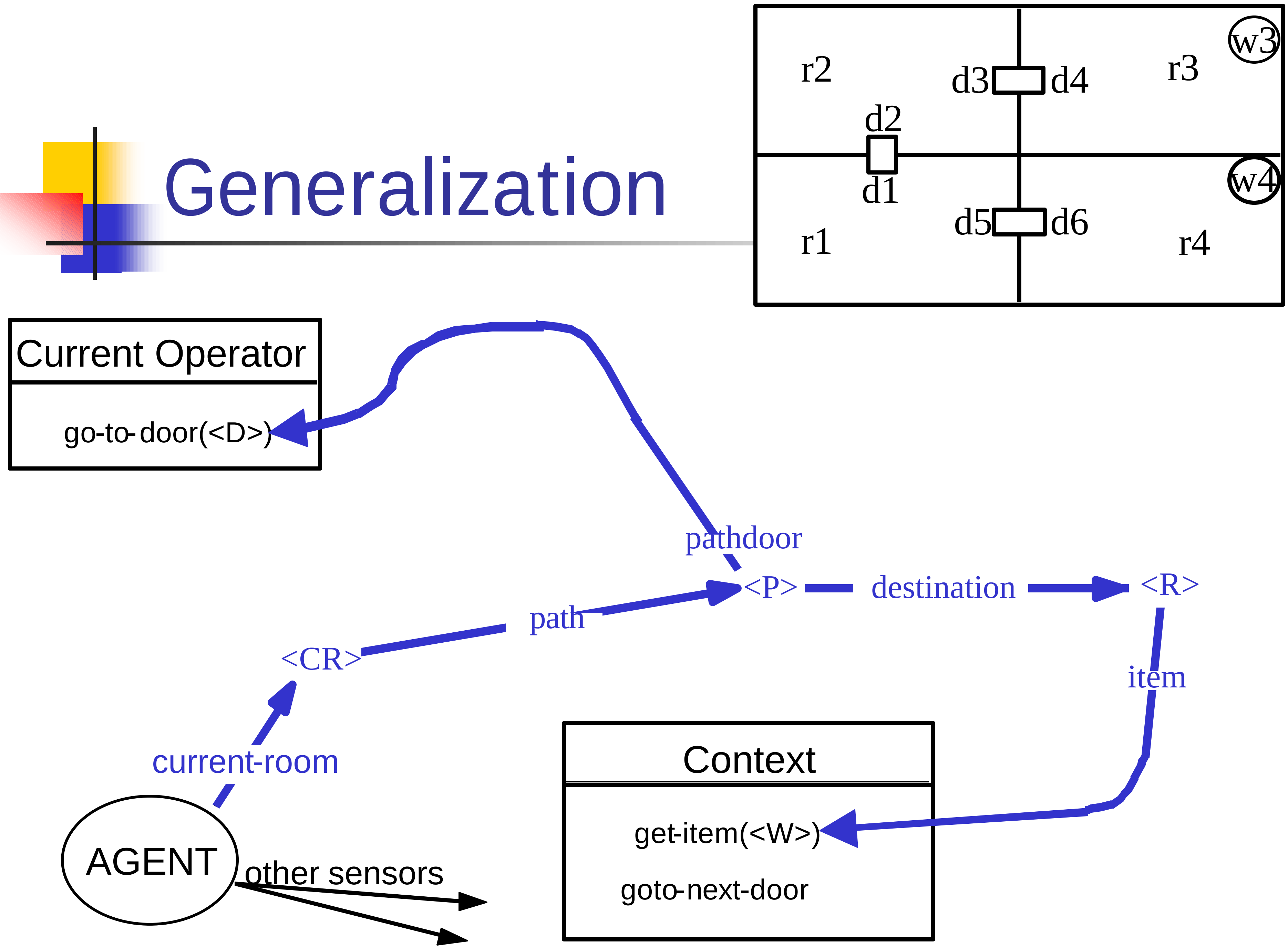


Another Refinement





Generalization



Background Knowledge in Learning

- Static Domain Knowledge

```
(r1 ^door d1 )
(r1 ^door d5 )
(r1 ^type room )
(r1 ^name room1 )
(r1 ^path p1 )
(r1 ^path p2 )
(r1 ^path p3 )
(p2 ^pathdoor d1)
...
```

Background Knowledge in Learning

- Dynamic Domain Knowledge

```
ADD:state1       (r1 ^item i1)
ADD:state2       (r1 ^item i2)
REMOVE:state5    (r1 ^item i1)
...
```

Background Knowledge

Internal Representation

```
CHANGES OF (r1 ^item)

state1:     {i1}
state2:     {i1,i2}
state5:     {i2}

...
```

```
VALUES OF (r1 ^item)

state1:     {i1}
state2:     {i1,i2}
state3:     {i1,i2}
state4:     {i1,i2}
state5:     {i2}
...
```

RULE:

item(+STATE, +Room -ListOfItems) ←

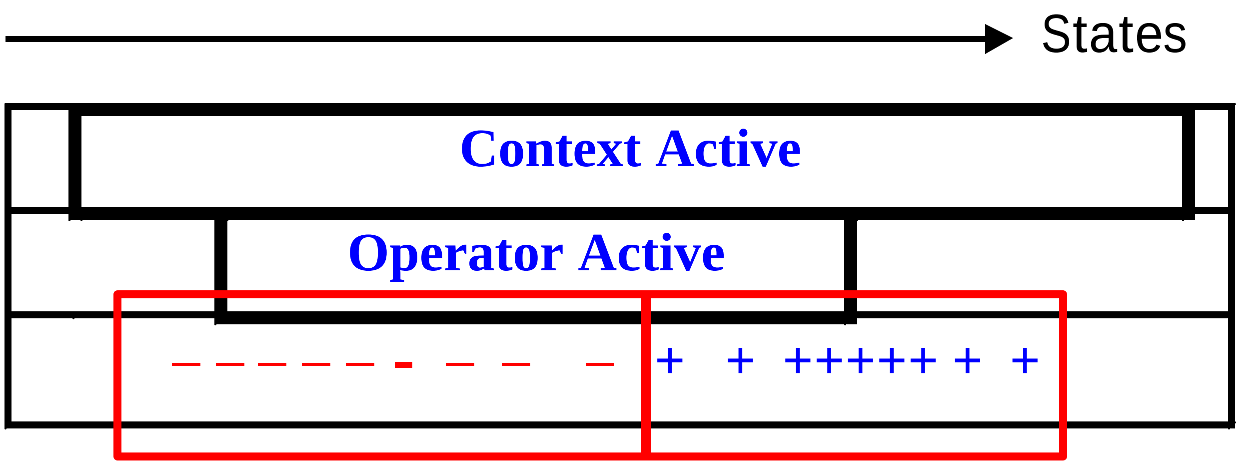
Find-last-change(STATE,ListOfItems).

Domain Knowledge
Representation

- Historical Queries (Episodic Database?)
  - Return all state intervals in your history when your x coordinate was larger than your y coordinate
  - Return all states when you have killed an enemy after you have picked-up an item at most 30-minutes ago
- Database contains changes not individual states:
  - Conditions that do not change frequently are checked at once for a set of cycles

Operator Concepts

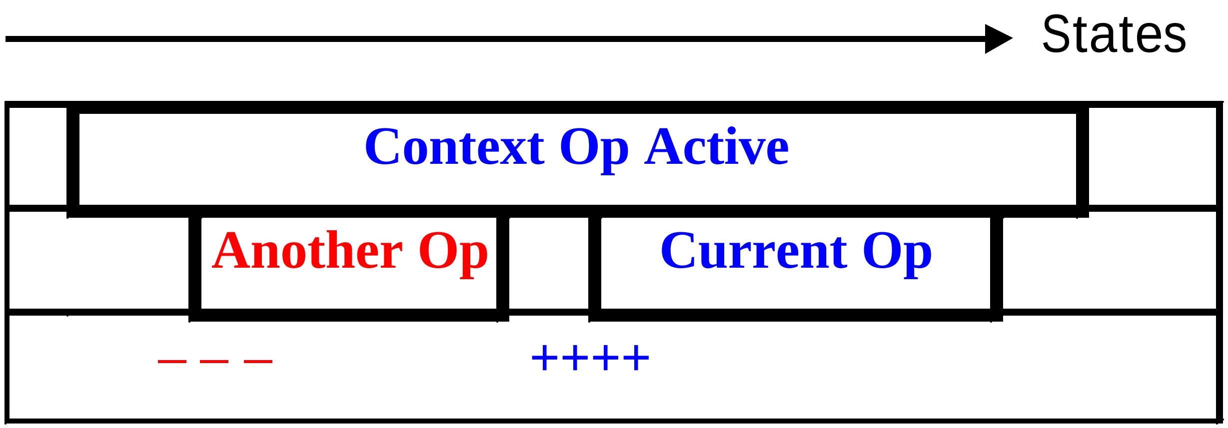
- Goal Condition Concept



- If true, do not select the operator
- if true and operator is active, finish the operator
- Not useful if another operator is active

Operator Concepts

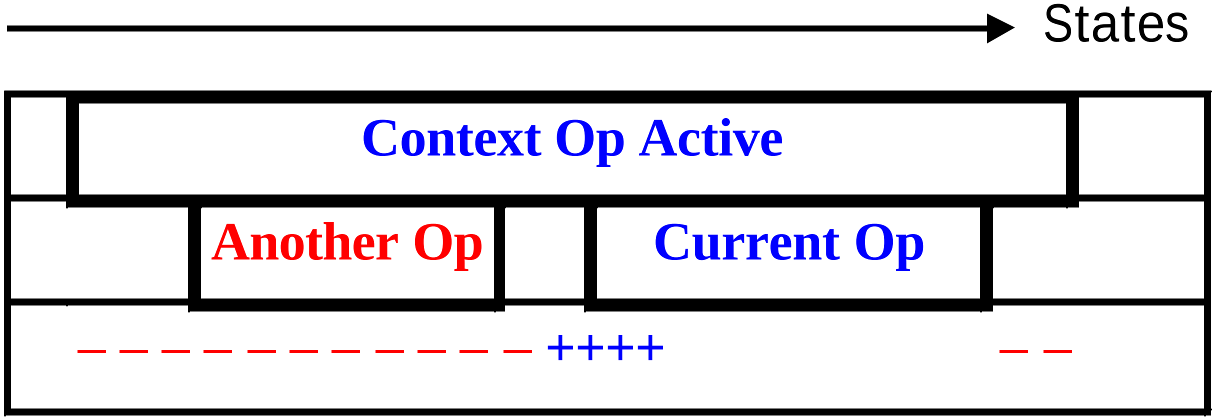
- Precondition Concept



- If false do not select the operator
- if True select the operator
- Not very useful except selection points
- few examples, if operators do not change frequently

Operator Concepts

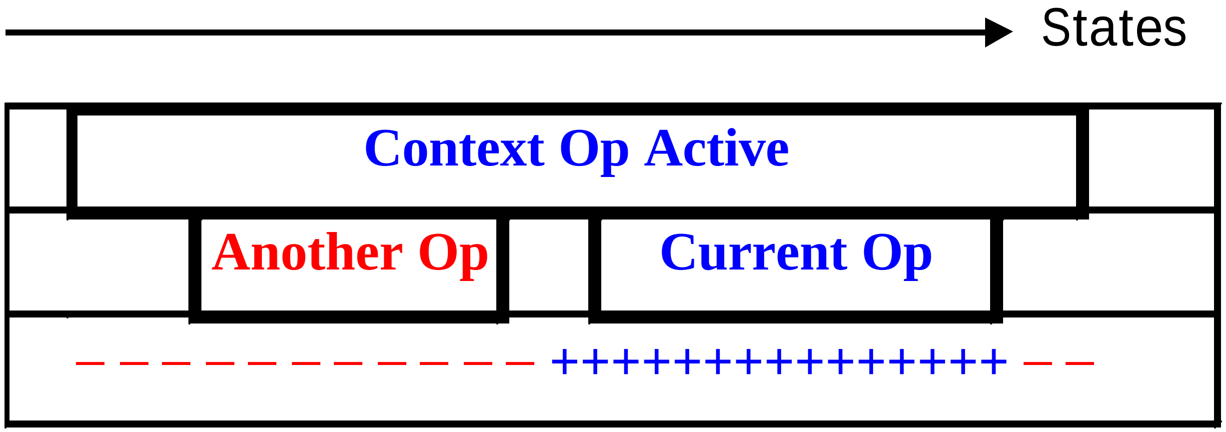
- Overriding-Precondition Concept



- if false, do not select the operator
- If true, select the operator
- If true, override another operator
- If false while already selected does not imply stop

Operator Concepts

- Condition Concept



- If false, do not select
- If false and already selected, stop
- If true, suggest overriding another operator

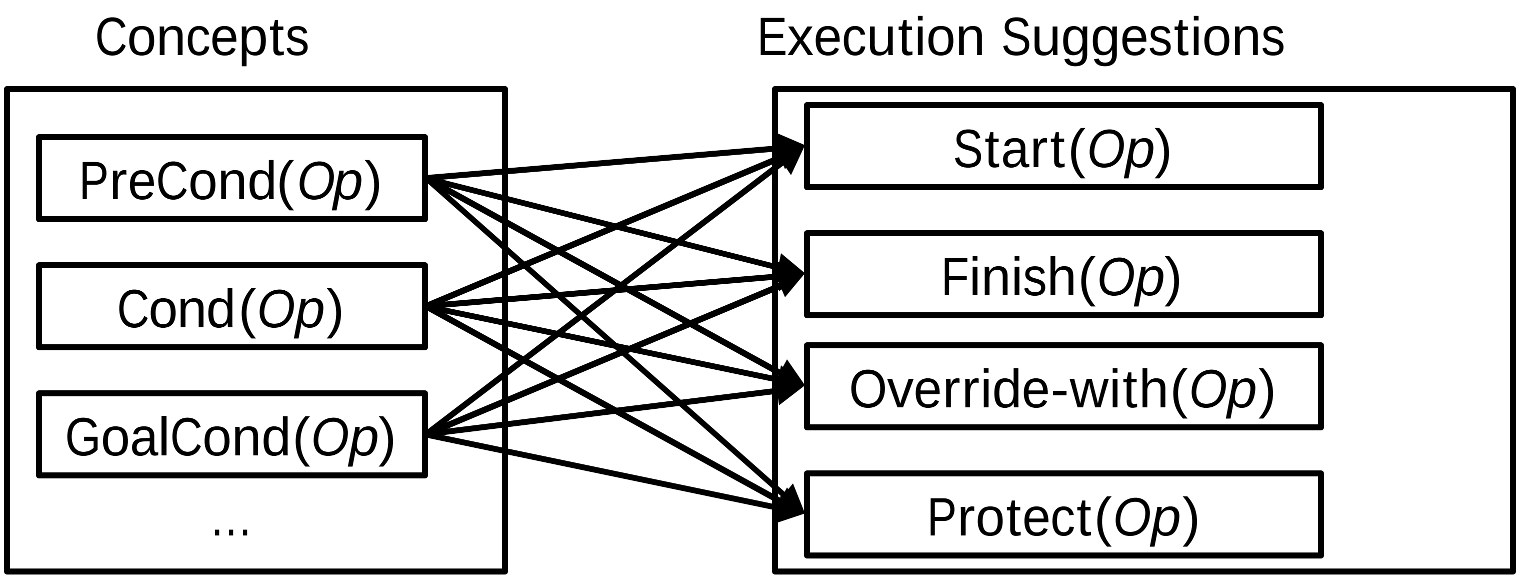
Using Concepts

- The accuracy of a concept may depend on
  - Properties of the domain
  - Properties of the operator
  - Available training data
- Testing examples may be used to predict accuracy of concept
- Priority to concepts that have higher accuracy for a specific operator

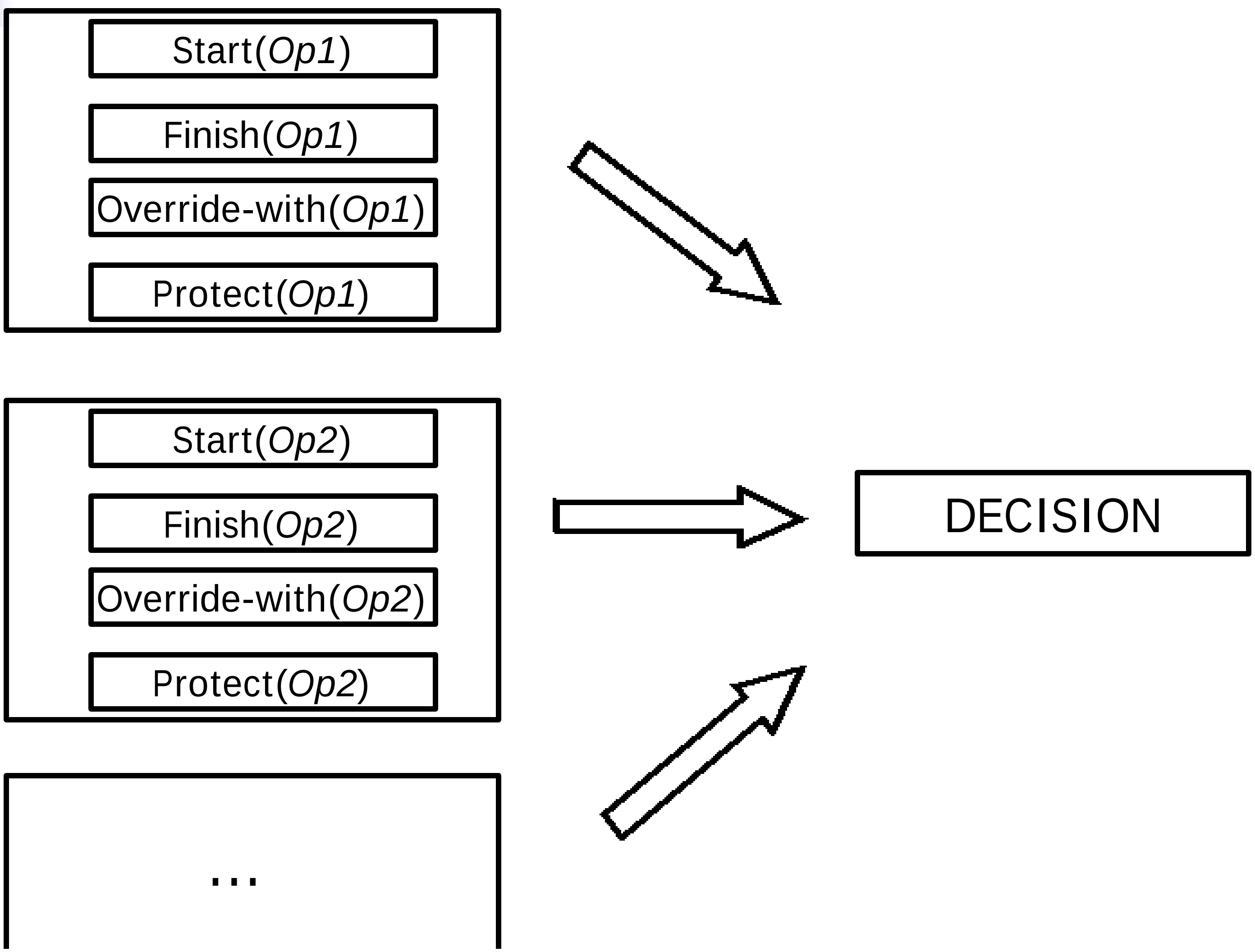
Using Concepts

- A *concept*(Op) has different implications if:
  - Op is currently active
  - The previous operator has just finished
  - Another operator is active
- Different concepts may have conflicting suggestions.
  - Operators (or suggestions of concepts of different operators) compete with each other

Using Concepts



Using Concepts



Using Concepts

- It is not trivial how much start/end/override/protection each concept suggest:
  - Only operator changes are observable
  - For example: Transition Op1-Op2
    - OP1 has finished and OP2 has started
    - OP2 has overridden the protection of OP1
  - A weight adjustment method can be used based on an external critic

Summary Predicates

- All aspects of the state are not observable
- The expert probably remembers a selective summary of the past, rather than using current state only to decide

- It is possible to have sensors that summarize past but it is not clear the past of which facts should be remembered.
- Some summary facts may be more commonly useful in lots of domains
  - achieved-before(Operator)
  - Previous-operator(Operator)

Summary Predicates

- achieved-5-times(Operator)
  (in the the intermediate context)

- The summaries should be efficiently implementable in Prolog and Soar

- They should help learning but should not provide too much degrees of freedom

Nugget and Coals

- Nuggets
  - ILP allows rich representation for domain knowledge (i.e. to encode task knowledge or common sense knowledge)
  - Can deal with rich structure in the sensors
  - Can use objects from high level operators

    (these are the important objects for the task)
  - Efficient testing of rules over the history of working memory → Episodic Memory?
- Coals
  - Not tested with a large domains yet
  - The number of required expert traces may be an important bottleneck

FUTURE WORK

- Experiment – Experiment - Experiment
- Incorporation of planning knowledge