Introduction to Intelligent, Knowledge-Based, and Cognitive Systems

89-674

Mor Vered
http://u.cs.biu.ac.il/~veredm/89-674/
mvered89230@gmail.com
Sunday 10-11, Building 202, Room 119
Operators

• Operators perform actions, either in the world or internally in the “mind” of an agent.

• Operators in a Tic Tac Toe program would mark X’s or O’s on the board. Operators for a soccer program would probably include move, turn, kick the ball, send messages to teammates, interpret messages from teammates, choose a tactic or strategy etc...

• Soar makes a distinction between the persistence of working memory elements created by operator application rules and the persistence of working memory elements created by other types of rules.

• Unlike i-support, this is a method of introducing permanence, o-support.

• Because they are the only way to make persistent changes to working memory, operators are the key to any Soar program.
Operators Cont.

• A rule is an *operator application rule* if it tests the selected operator and modifies the state.

• Soar makes a distinction between knowledge that modifies the current state (knowledge in *operator applications*), and knowledge that just computes the entailments of the current situation (including which operators should be considered for the current state).

• *Operator application* needs to create persistent results – otherwise will be an infinite loop.

• All other rules in Soar compute the entailments/elaborations of the current state without actually changing it.
Operators

- Soar’s basic operation is a cycle in which operators are continually proposed, selected, and applied.
- Rules fire to propose and apply operators
- The decision procedure selects the current operator
- Separates where decisions are being made (proposal and selection) and where actions are being performed (application)
Operators

• **Propose** the operator (IF condition-x THEN operator=foo)
• **Apply** the operator (IF operator=foo THEN do things to working memory)
• **Terminate** the operator (IF not-x THEN reconsider operator=foo)
Operators

• Operators must first be created in working memory by *proposal* rules.
• Proposal rules test features of the state to ensure that the operator is appropriate and then they create a representation of the operator in working memory along with an *acceptable preference* for that operator.
• For simple operators, the representation in working memory is often just a name of the operator (“Hello-World”).
• More complex operators can also contain representations of parameters.
• Once the operator is *selected*, rules that apply the operator will match and perform whatever actions are appropriate by creating or removing working memory elements.
Phases of the Soar decision cycle

- **Input** (insert new WMEs from input-link)
- Operator **proposal** rules, elaborations
  - Rules that interpret input, add WMEs
  - Rules that propose new operator WME
    - (state <s> ^operator <o> + )
- **Decision** on selected operator
- Operator **application**
  - Rules with o-support (state <s> ^operator.name)
- **Output**: Send to IO output-link
Operator-related phases

• **Proposal:** Rules that propose WMEs for operator
  - i-supported (<s> ^operator <o> +)
  - Allow reactivity, response to inputs

• **Selection:**

• **Application:**
  - Rules that modify WMEs based on operator
    - o-supported rules: test (<s> ^operator.name)
    - Permanent WME changes
    - Elaborations (i-supported rules)

• **Termination:**
Hello World Operator

• For printing “Hello World”, an operator is not necessary because there are no alternatives, but as soon as other actions are possible and choices must be made, operators become necessary.

• To use an operator, we need two rules: one to propose the operator and one to apply it:

Propose*hello-world:
If I exist, propose the hello-world operator.

Apply*hello-world:
If the hello-world operator is selected, write “Hello World” and halt.
Hello World Operator - Propose

• “propose*hello-world” proposes the hello-world operator.
• A rule proposes an operator by creating an acceptable preference for the operator.
• An acceptable preference is a statement that an operator is as a candidate for selection.
Hello World Operator

• The first action creates an acceptable preference for a new operator (which is added to working memory)

• The second action creates a working memory element that augments the operator with its name.

• A preference looks just like other working memory elements except that it has a fourth item: the type of preference, in this case is “+”.

```
sp {propose*hello-world
   (state <s> ^type state)
   --
   (<s> ^operator <o> +)
   (<o> ^name hello-world})
```

<s> in the action is replaced by the identifier matched by <s> in the condition

{o> is new in the action and is replaced by a new, unique identifier

+ indicates that this is an acceptable preference

{o> is replaced by the same identifier in all actions
Hello World Operator

• The value of the preference, <o>, is a new WME.

• Soar automatically creates a new identifier and uses it for all occurrences of that variable in the action. For example, if o1 is the identifier created for <o>, then (s1^operator o1 +) and (o1 ^name hello-world) are added to working memory.
Hello World Operator - Apply

• Once the operator is selected, apply*hello-world should fire.

• If the same variable is used in multiple rules, it can match completely different identifiers or constants – that is, the identity of variables only matters within a rule.

• As a general convention, <s> is usually used to match state identifiers and <o> is usually used to match operator identifiers.
“Hello, World” Operator: Apply

- If the currently selected operator is hello
- Then write “hello, world!”

```
sp {apply*hello*1
  (state <s> ^operator.name hello)
  --(×o> ^name hello)
  --(write |Hello, World!|)
  }(write |Hello, World!|)
}
Adjusting Working Elements

• To replace a value in Soar, you must remove the original working memory element and create a new one.

• There is no way to just replace the value field in an existing working memory element.

• You cannot modify a working memory element once it is created.

• To delete a WME, you specify the working memory element in the action but with a minus “-“ at the end.

\[
<j> \; ^\text{contents} \; 0
\]

\[
<j> \; ^\text{contents} \; \langle\text{contents}\rangle \; -)
\]
Homework

- Re-do hello world
  - Write a hello-world.soar rule file
  - “source hello-world.soar” in soar
  - Play with debug levels to see what’s going on
- Create new Hello World program that modifies a state count by increasing its value. Count will start at zero and will increase by one for every cycle. For every cycle print to the screen “Hello World number count : “
  - Deadline 4/03/2017
  - Submit by mail : mvered89230@gmail.com