The Engineer’s Guide to Soar
Course 01: Soar Essentials

Project 04: Output

By Dr. Bryan Stearns, 2024

Problem

Our agent selects suppliers in a particular order, but it does not send any of its results to output.

Solution

After the agent has evaluated all suppliers in order, copy the evaluated suppliers to the agent's output-link structure.

- The output-link is provided automatically by Soar to support output to the environment.
- Rather than add to the output-link incrementally, we copy final results to the output link all at once so that the environment receives everything in a single output message.
- (We will address the problem of preserving supplier order in the output message in the next project.)

Project Goal

We want our agent to attach selected suppliers to the output link.
The resulting output-link structure should be as follows:

```
S1 ^io.output-link I3
    ^recommended-supplier |supplier05|
    ^recommended-supplier |supplier03|
    ^recommended-supplier |supplier01|
    ^recommended-supplier |supplier04|
    ^recommended-supplier |supplier06|
    ^recommended-supplier |supplier02|
```

Lesson 04 – Outline

This lesson explains the following new concepts:

1. Using the `output-link` structure
2. Adding *unary* preferences when proposing an operator
   - Acceptable (+)
   - Worst (<)
   - Indifferent (=)
3. Using the `(interrupt)` RHS function to pause the agent

The Output Link

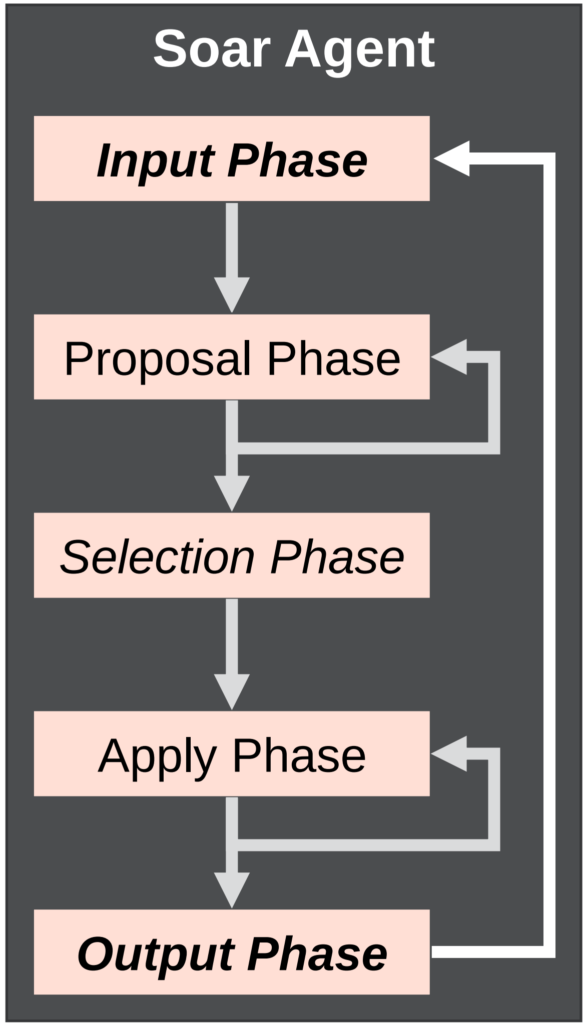
Communicating with the Environment

Using the Output Link

Soar provides the output-link for sending data to the environment.

- Any time the agent modifies content under the output-link ID, the updated structure will be sent to the environment at the end of the current decision cycle, during the *Output Phase*.
- The environment code will be notified of an update event any time the Soar agent sends output-link updates.
  - So it's good practice to only send fully-formed output that is ready to be read by the environment.
  - *Don't* gradually construct agent output on the output-link in stages

```
S1 ^io I1
    ^input-link I2
        (input comes in here)
    ^output-link I3
        (output here is sent out)
```



Making an Output Operator

We will add another operator to our existing code from Project 03

- After all `(select-supplier)` operators have finished, then send the selected suppliers to the output-link all at once

The logic for our new operator is as follows:

- Propose the `(output-suppliers)` operator any time the agent knows it has selected at least one supplier
- Select the `(output-suppliers)` operator only if there are no `(select-supplier)` operators still proposed
  - (That is, give the `(output-suppliers)` operator *worst* (“<“) preference.)
- Apply the `(output-suppliers)` operator by copying all `(<s> ^selected <sup>)` structures to the output-link.

Let’s Write Some Code!

1. Open your `agent_starter.soar` file and look at the first rule there.
2. Follow the “STEP 1” instructions to fill in the basic rule code.
3. Proceed to the next slide when you are ready to tackle “STEP 2”.
   - We still need to give preferences to `(output-suppliers)` so that we don’t make a tie impasse with `(select-supplier)` operators.
   - But we don’t need to write a separate preference rule.
   - We can do this directly from our proposal rule!

Proposing with Unary Preferences

Communicating with the Environment

Unary Preferences – Review

Lesson 03 introduced preferences. There are two types of preferences:

1. Binary preferences relate two specific proposed operators to each other
2. Unary preferences tag a single proposed operator
   - The agent automatically compares all unary preferences of all proposed operators

The unary preferences are shown again below:

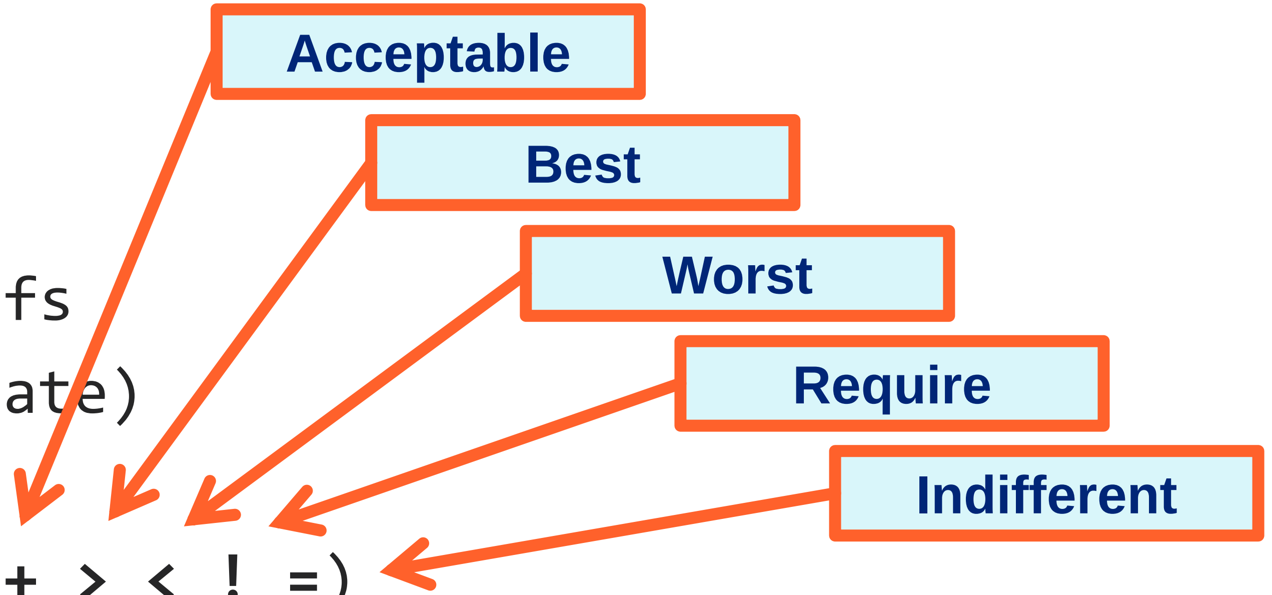
Preference Name	Syntax	Description
`Acceptable`	+	States that a value is a candidate for selection
`Reject`	–	Removes the value as a candidate for selection
`Best`	>	If a value is best, it will be selected over any other value that is not also best (or required)
`Worst`	<	States that the value should be selected only if there are no alternatives
`Unary Indifferent`	=	When two or more competing values both have unary indifferent preferences, choose one randomly
`Numeric-Indifferent`	= number	Like unary indifferent, but the number weights the chance of the operator's selection relative to others
`Require`	!	States that the value must be selected (preferred over all others)
`Prohibit`	~	States that the value cannot be selected

Proposing with Unary Preferences

When you propose an using the “+” symbol, that “+” is a unary preference!

You can attach *any number* of unary preferences when you propose operators.

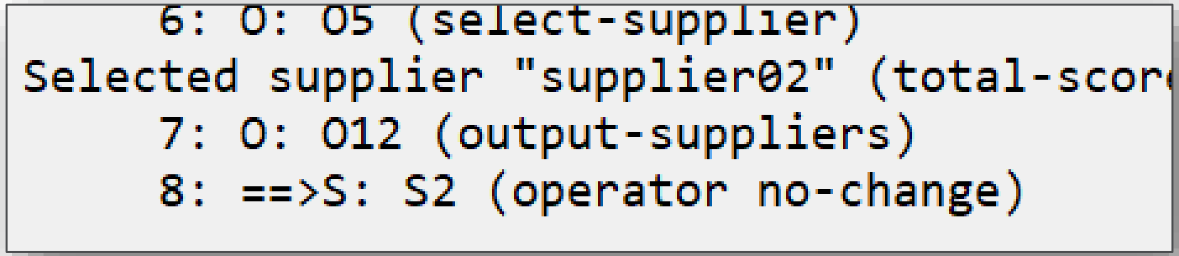
```
sp {propose*all-the-prefs
    (state <s> ^type state)
    -->
    (<s> ^operator <o> + > < ! =)
    (<o> ^name |all-the-prefs!|)}
```



If Best and Worst are combined, the agent evaluates Best first.

Let’s Write Some Code!

1. Reopen your `agent_starter.soar` file to edit the proposal rule.
2. Follow the “STEP 2” instructions to give “worst” and “indifferent” preferences to the `(output-suppliers)` operator.
   - “worst” so that it is selected after `(select-suppliers)`
   - “indifferent” because multiple `(output-suppliers)` operators will be proposed (one for each `^selected` object)
3. Run the agent for 8 steps
   - You should see this →



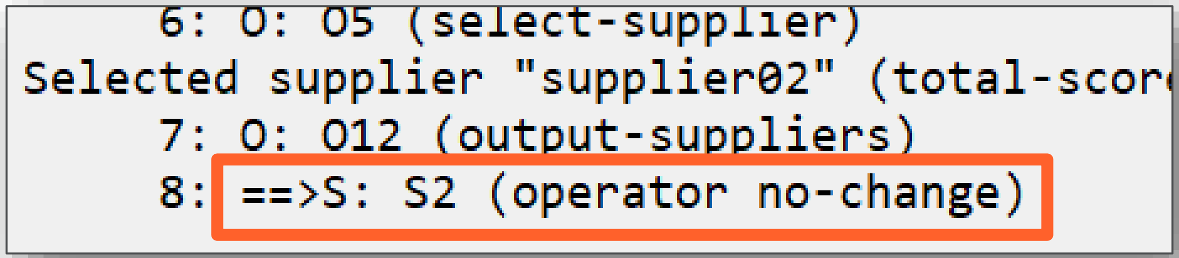
```
     6: O: O5 (select-supplier)
Selected supplier "supplier02" (total-scor
     7: O: O12 (output-suppliers)
     8: ==>S: S2 (operator no-change)
```

4. Proceed to the next slide when you are done.

Using the `(interrupt)` RHS Function

To Stop the Agent When it’s Work is Done

Applying `(output-suppliers)`



```
     6: O: O5 (select-supplier)
Selected supplier "supplier02" (total-scor
     7: O: O12 (output-suppliers)
     8: ==>S: S2 (operator no-change)
```

We're almost done!

We need to write an apply rule to avoid the `(operator no-change)` impasse.

The rule should fire once for every `^selected` WME on the state.

We want the apply rule to do 4 things:

1. **Copy** recommended suppliers to the output-link.
2. **Remove** the copied supplier object from the state
   - This will cause the proposal to retract when the work is done.
3. **Write** a message to show what the agent recommended in its output
4. **Pause** the agent
   - It is done once it sends output to the environment
   - To pause the agent, we can use the (**`interrupt`**) RHS Function!

The `(interrupt)` and `(halt)` RHS Functions

The (**interrupt**) RHS Function pauses Soar as soon as it fires the rule.

- This does not stop rule actions that occur at the same time as the interrupt.
- This does not abort the agent. It is only paused.
- You can resume the agent by using the “`step`” or “`run`” commands.

The (**halt**) RHS Function is like `(interrupt)`, but it aborts the Soar agent

- The agent cannot resume. It is terminated.

```
sp {elaborate*interrupting-cow
    (state <s> ^io.input-link.who-is-there |cow|)
    -->
    (write |MOO!|)
    (interrupt) }
```

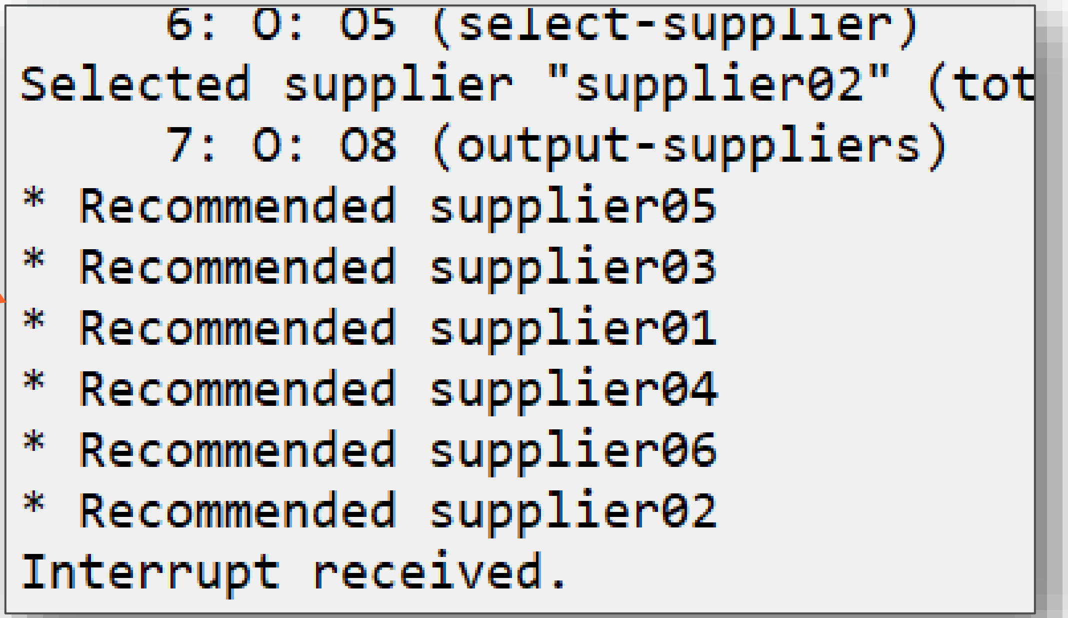
Let’s Write Some Code!

1. Uncomment the apply rule code in your `agent_starter.soar` file.
2. Start by making the condition for the selected `(output-suppliers)` operator.
   - Remember: It’s not an apply rule unless it tests for the selected operator.
   - Feel free to reference your apply rule code from Lesson 03.
3. Fill in the four underscore sections on the RHS.
4. Fill out the LHS to supply variables that you find you need for the RHS.
5. To test your code, use ”`run`” rather than “`step`”.
   - Your `(interrupt)` RHS Function should stop the agent.
6. If you get stuck, compare your code with `instructor_solution.soar`.
7. Proceed to the next slide when you are done.

Final Results

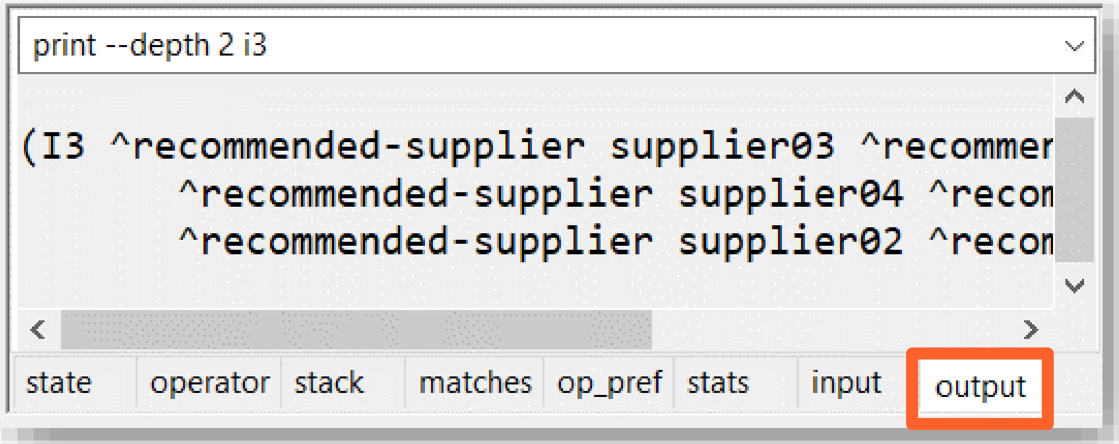
After running your agent, you should see the following:

- Each of the 6 input suppliers is printed
- (Order doesn't matter. The apply rules fire in parallel.)



```
      6: O: O5 (select-supplier)
Selected supplier "supplier02" (tot
      7: O: O8 (output-suppliers)
* Recommended supplier05
* Recommended supplier03
* Recommended supplier01
* Recommended supplier04
* Recommended supplier06
* Recommended supplier02
Interrupt received.
```

You should be able to see the output appear in the output auto-update tab in the debugger, as shown.



```
print --depth 2 i3

(I3 ^recommended-supplier supplier03 ^recommer
       ^recommended-supplier supplier04 ^recon
       ^recommended-supplier supplier02 ^recon
```

If so, **congratulations**!
You have completed Project 04!