

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

Project 02: **The Input Link**

By Dr. Bryan Stearns, 2024



Our Tutorial Task

We are building a “Supplier Sorting” agent

- It should sort input items according to various configurable properties.
- *(Review “ProblemOverview.pdf” in the course directory for a description of the task we are solving.)*

Problem

Our agent does not yet do anything relevant to the task.

Solution

Start by parsing some task input!

- Print suppliers and weights to std-out
- (validates that the agent is reading inputs correctly)

Lesson 02 – Outline

This lesson explains the following new concepts:

1. Soar dot-notation syntax
2. Using the `^input-link` structure to receive input from the environment
3. Loading a file using CLI commands
4. Firing multiple instances of a single rule at once
5. Using the `matches` CLI command
6. Using variables for attributes

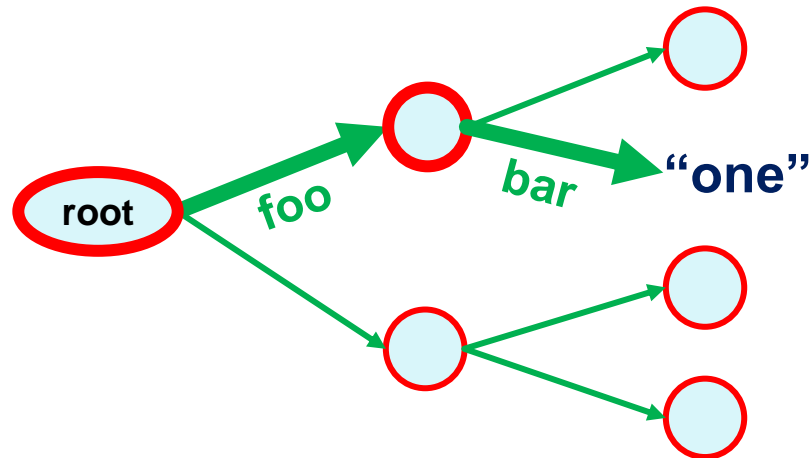
Soar Dot-Notation Syntax

A simpler way to code

Explicit Notation

We've seen how to write conditions that test a chain of edges in the WM graph...

```
sp {example*rule  
  (state <s> ^foo <n>)  
  (<n> ^bar |one|)  
  -->  
  (... ) }
```



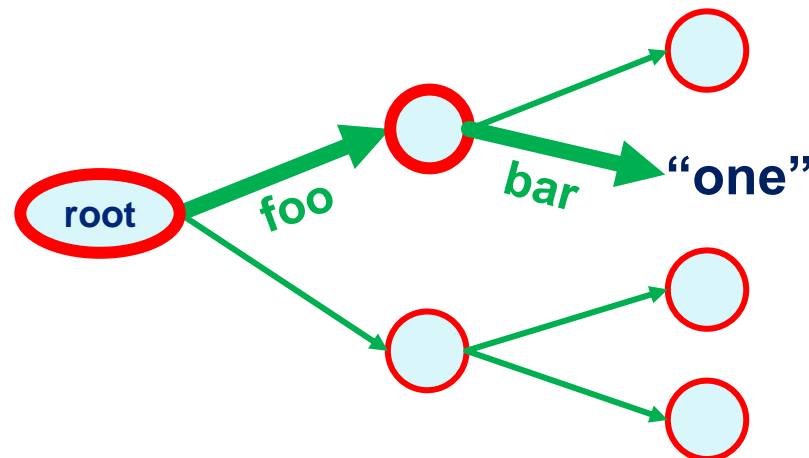
Dot Notation

Soar allows a shorthand syntax called **dot-notation**!

- Chain attribute names with a dot in between
- You can make the chain as long as you like!

```
sp {example*rule  
  (state <s> ^foo.bar |one|)  
  -->  
  (... ) }
```

This is equivalent to the rule on the previous slide.



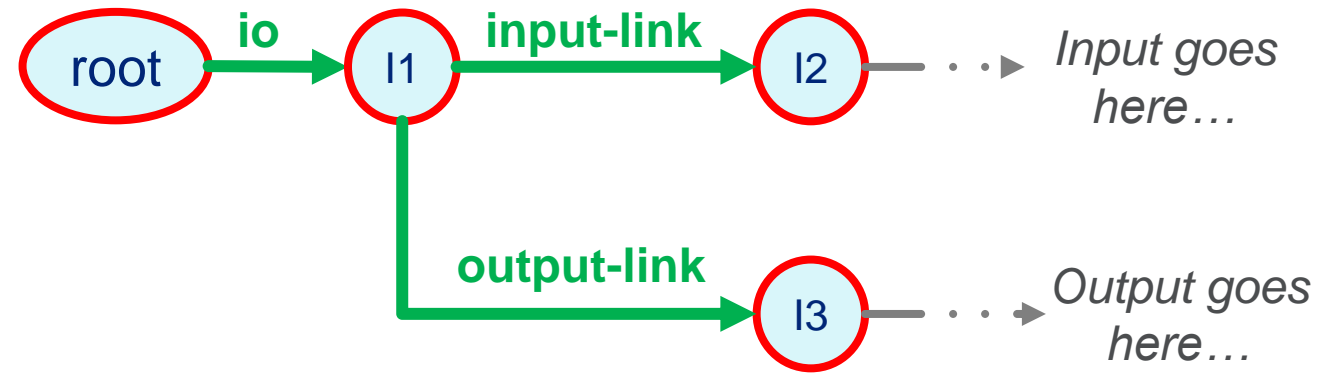
Using the Input Link

To receive input from the environment

The io Structure

Soar provides the WMEs shown on the right

- **io.input-link**
 - Any input sent to Soar appears here
 - Input must be created by an outside program that interfaces with Soar
- **io.output-link**
 - The agent puts WMEs here to send them to the environment
 - Output must be interpreted by an outside program that interfaces with Soar



Project Input

This is a text notation for WM content.
Node IDs are a letter+number combo. ("P1")

For this course, default input is provided for your project agents

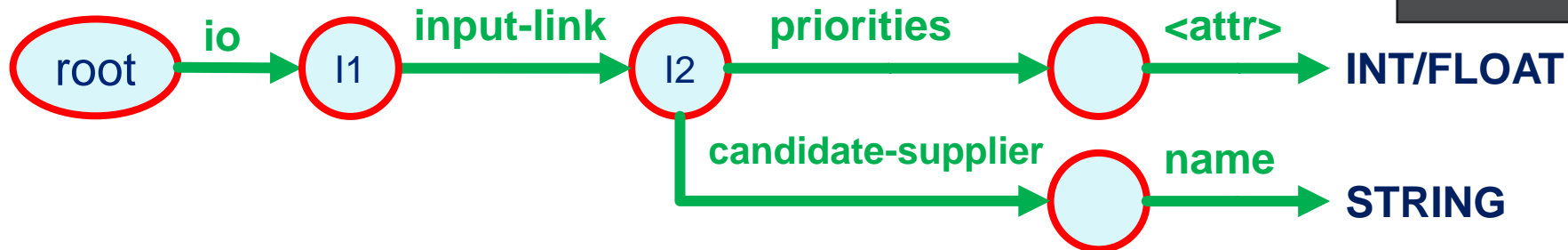
- We'll see how to enable this input later in this lesson.

The input schema we use for Project 02 is:

- `^input-link.priorities.<attr>` INT/FLOAT
 - Multiple edges match `<attr>`, such as "quality", "speed", etc.
- `^input-link.candidate-supplier.name` STRING
 - There are 6 named suppliers included in this input

```
S1 ^io.input-link I2
    ^priorities P1
        ^total-cost 11.01
        ^sustainability 11
        ^quality 11
        ^availability 8
        ^packaging 8
        ^speed 7

    ^candidate-supplier C1
        ^name |supplier01|
    ^candidate-supplier C2
        ^name |supplier02|
    ...
    ^candidate-supplier C6
        ^name |supplier06|
```



Loading a File Using CLI Commands

So we can load our test input

Using Fake Input

For this lesson, we do not have an external environment application sending input or reading output.

We will instead load a `.soar` file that simulates input.

- The file is “**fake_agent_input.soar**”, found in the main tutorial directory.
- It defines rules that create WMEs on the input-link.
 - Normally you should only let the environment modify the input-link structures!
 - But this approach is useful for testing during agent development.
- You do not need to worry about these rules, but feel free to examine the file content if desired.

Loading a Soar File

The CLI command to load a file into Soar is simply:

- `load file [filename]`

An alias/shorthand for this same command is:

- `source [filename]`

This command will cause Soar to open the indicated file and read all its contents as if they were typed into the Soar CLI.

Loading Test Input

The top of your `agent_starter.soar` file includes two lines that automatically load our test input for you:

```
cd ".."  
load file "fake_agent_input.soar"
```

The first line changes the working directory from the lesson directory to the main tutorial directory.

The second line loads the `fake_agent_input.soar` file that is found there.

Filesystem Commands

Soar includes various CLI commands for interacting with the filesystem:

- **cd**: Change directory (same usage as in normal OS terminals)
- **load [filename]**: Load the contents of the indicated file
- **ls**: List the current directory contents (same usage as in normal OS terminals)
- **pwd**: Print the current working directory path
- **pushd [dirname]**: Enter the indicated directory within the current directory
- **popd**: Exit the most recently pushed directory level
- **dirs**: Print the current directory stack

Firing Multiple Instances of a Soar Rule at Once

With some coding!

Using Our Input

When we load our test input file, the agent sees input as shown to the right.

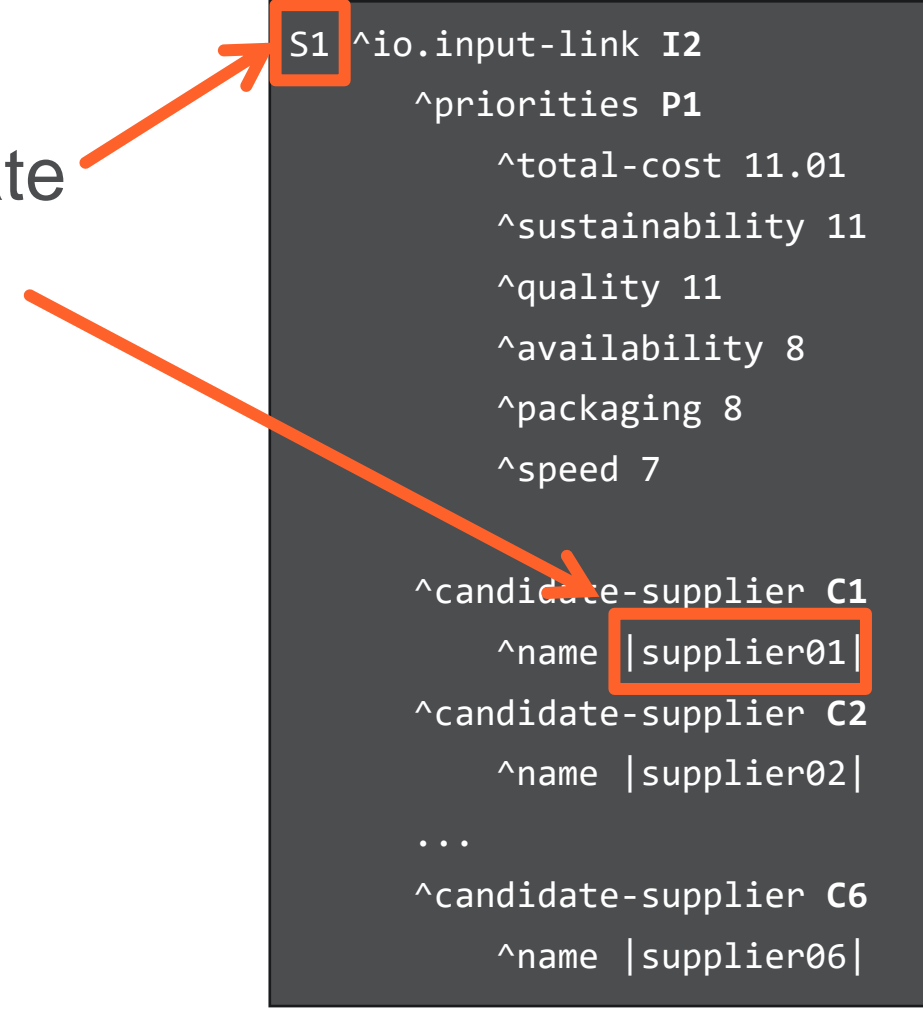
We first write a rule that prints off the name of each input supplier.

```
S1 ^io.input-link I2
    ^priorities P1
        ^total-cost 11.01
        ^sustainability 11
        ^quality 11
        ^availability 8
        ^packaging 8
        ^speed 7

    ^candidate-supplier C1
        ^name |supplier01|
    ^candidate-supplier C2
        ^name |supplier02|
    ...
    ^candidate-supplier C6
        ^name |supplier06|
```

Using Our Input

What is the attribute path from the root state node (labeled S1) to any supplier's name?



The diagram shows a root state node labeled S1, which is highlighted with an orange box. Two orange arrows originate from the text 'What is the attribute path from the root state node (labeled S1) to any supplier's name?'. One arrow points to the S1 node, and the other points to the 'supplier01' value in the 'name' attribute of the 'candidate-supplier C1' node. The JSON structure is as follows:

```
S1 ^io.input-link I2
  ^priorities P1
    ^total-cost 11.01
    ^sustainability 11
    ^quality 11
    ^availability 8
    ^packaging 8
    ^speed 7
  ^candidate-supplier C1
    ^name |supplier01|
  ^candidate-supplier C2
    ^name |supplier02|
  ...
  ^candidate-supplier C6
    ^name |supplier06|
```

Let's Write Some Code!

1. Open your `agent_starter.soar` file for Project02.
2. Find the provided rule starter named: `"elaborate*suppliersort-main*input*supplier-name"`
3. Uncomment this rule. (Remove the leading `"#"` characters.)
4. Fill in the LHS underscores with the dot-notation path to the input supplier name
5. Fill in the RHS underscores with the supplier name variable that you referenced on the LHS.
6. Add a newline character to the end of the rule's output message.
7. Proceed to the next slide for the solution when you are done.

```
sp {elaborate*suppliersort-main*input*supplier-name
    "Print any input candidate supplier names"
    (state <s> ^___ <sup-name>)
    -->
    (write |INPUT: Supplier candidate: | ___)}
```

Let's Write Some Code!

1. Here is the completed rule
2. Run the agent with this rule for 1 step and see what happens!
3. Proceed to the next slide when you're done.


```
sp {elaborate*suppliersort-main*input*supplier-name
  "Print any input candidate supplier names"
  (state <s> ^io.input-link.candidate-supplier.name <sup-name>)
  -->
  (write |INPUT: Supplier candidate: | <sup-name> (crLf))}
```

What Happened?

You should have seen this output

The agent printed ***every supplier name***

- Though we only coded 1 rule



```
**
*
Total: 3 productions sourced.
step
INPUT: Supplier candidate: supplier06
INPUT: Supplier candidate: supplier05
INPUT: Supplier candidate: supplier04
INPUT: Supplier candidate: supplier03
INPUT: Supplier candidate: supplier02
INPUT: Supplier candidate: supplier01
      1: ==>S: S2 (state no-change)
```

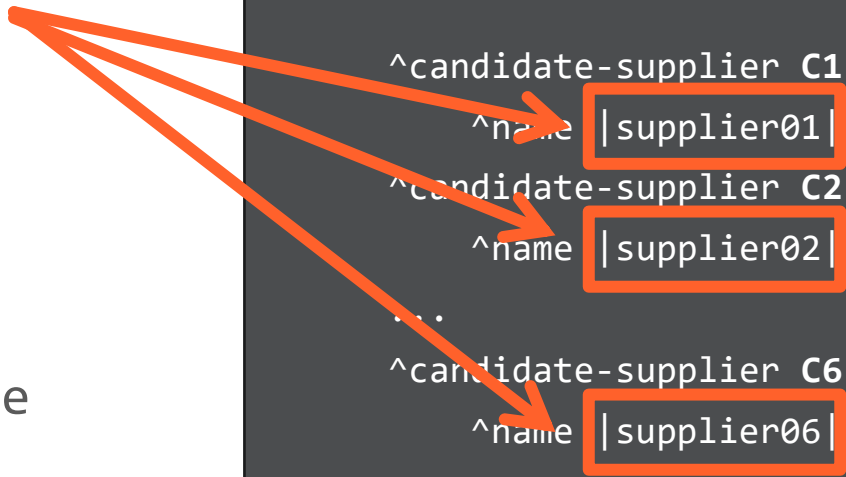
```
sp {elaborate*suppliersort-main*input*supplier-name
    "Print any input candidate supplier names"
    (state <s> ^io.input-link.candidate-supplier.name <sup-name>)
    -->
    (write |INPUT: Supplier candidate: | <sup-name> (crLf))}
```

What Happened?

Soar rules fire multiple times for **every combination of variable bindings**.

- There are 6 ways that the WM graph can provide a value for **<sup-name>**
- So the rule fires 6 times, once for each value
 - In no particular order

```
sp {elaborate*suppliersort-main*input*supplier-name
  "Print any input candidate supplier names"
  (state <s> ^io.input-link.candidate-supplier.name <sup-name>)
-->
(write |INPUT: Supplier candidate: | <sup-name> (crLf))}
```



```
S1 ^io.input-link I2
  ^priorities P1
    ^total-cost 11.01
    ^sustainability 11
    ^quality 11
    ^availability 8
    ^packaging 8
    ^speed 7
  ^candidate-supplier C1
    ^name |supplier01|
  ^candidate-supplier C2
    ^name |supplier02|
  ...
  ^candidate-supplier C6
    ^name |supplier06|
```

Instantiations

A rule **instantiation** in Soar is a rule firing for a particular set of variable bindings.

```
sp {elaborate*suppliersort-main*input*supplier-name
    "Print any input candidate supplier names"
    (state <s> ^io.input-link.candidate-supplier.name <sup-name>)
    -->
    (write |INPUT: Supplier candidate: | <sup-name> (crLf))}
```

RULE

INSTANTIATIONS

{instantiation1

```
(S1 ^io I1)
(I1 ^input-link I2)
(I2 ^candidate-supplier C1)
(C1 ^name |supplier01|)
-->
(write |INPUT: Supplier candidate: | |supplier01| (crLf))}
```

Dot notation is expanded to see how each ID is instantiated.

{instantiation2

```
(S1 ^io I1)
(I1 ^input-link I2)
(I2 ^candidate-supplier C2)
(C2 ^name |supplier02|)
-->
(write |INPUT: Supplier candidate: | |supplier02| (crLf))}
```

Notice both the candidate-supplier ID and the name are different for each instantiation.

Using the matches CLI Command

For Debugging Soar Rule Firings

Seeing the Number of Matches

Soar provides the `matches` CLI command to let you see the different matches of a rule. After your agent prints its output, enter the following command into the CLI input area:

- `matches elaborate*suppliersort-main*input*supplier-name`

You should see the output shown below.

- It shows the expanded conditions of our rule and how many WMEs match each condition component

The leading number shows how many WMEs match the WME pattern of that line in the rule.

Notice that there is only 1 input-link edge, but there are 6 candidate-supplier edges, each with 1 name for a total of 6 name edges.

```
matches elaborate*suppliersort-main*input*supplier-name
1 (state <s> ^io <i*1>)
1 (<i*1> ^input-link <i*2>)
6 (<i*2> ^candidate-supplier <c*1>)
6 (<c*1> ^name <sup-name>)

6 complete matches.
```

6 instantiations

Seeing the Matches

If you add the “-w” / “--wmes” command option, Soar will print **each match** of the rule you are inspecting

- `matches -w elaborate*suppliersort-main*input*supplier-name`

This is a very useful command when debugging your rules!

The printout shows each set of WM elements that together satisfy the rule conditions in each instantiation.

```
6 complete matches.
*** Complete Matches ***
(11: S1 ^io I1)
(12: I1 ^input-link I2)
(14: I2 ^candidate-supplier C8)
(65: C8 ^name supplier06)

(11: S1 ^io I1)
(12: I1 ^input-link I2)
(15: I2 ^candidate-supplier C7)
(56: C7 ^name supplier05)

(11: S1 ^io I1)
(12: I1 ^input-link I2)
(16: I2 ^candidate-supplier C6)
```

Using Variables For Attributes

Finishing Up Our Project

Printing Priorities

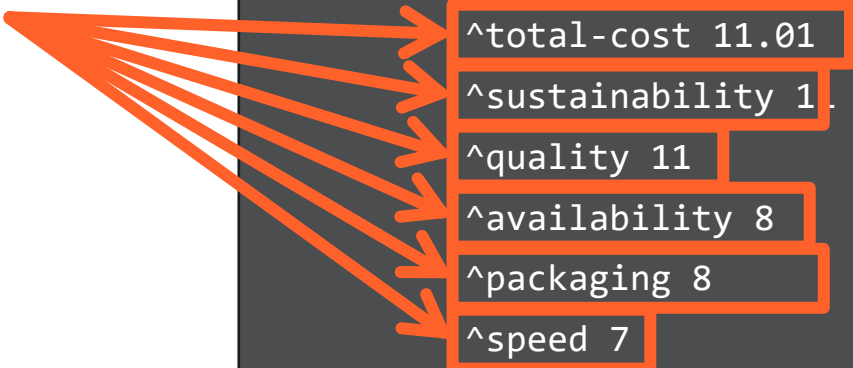
We also want our agent to print the names of each priority attribute with its weight value.

We could make a rule for each attribute path, such as these:

```
sp {elaborate*suppliersort-main*input*config-priorities*total-cost
  (state <s> ^io.input-link.priorities.total-cost <weight>)
  -->
  (write |INPUT WEIGHT: total-cost:| <weight> (crlf))}
```

```
sp {elaborate*suppliersort-main*input*config-priorities*quality
  (state <s> ^io.input-link.priorities.quality <weight>)
  -->
  (write |INPUT WEIGHT: quality:| <weight> (crlf))}
```

But there's an easier way!



```
S1 ^io.input-link I2
  ^priorities P1
    ^total-cost 11.01
    ^sustainability 1
    ^quality 11
    ^availability 8
    ^packaging 8
    ^speed 7
  ^candidate-supplier C1
    ^name |supplier01|
  ^candidate-supplier C2
    ^name |supplier02|
  ...
  ^candidate-supplier C6
    ^name |supplier06|
```

Attribute Variables

Variables can be used for attributes as well as for IDs and values.

The single rule below will match on *every* attribute:value pair underneath the priorities input object.

```
sp {elaborate*suppliersort-main*input*config-priorities
    (state <s> ^io.input-link.priorities.<attr> <weight>)
    -->
    (write |INPUT WEIGHT: | <attr> |: | <weight> (crlf))}
```

```
S1 ^io.input-link I2
    ^priorities P1
        ^total-cost 11.01
        ^sustainability 11
        ^quality 11
        ^availability 8
        ^packaging 8
        ^speed 7

        ^candidate-supplier C1
            ^name |supplier01|
        ^candidate-supplier C2
            ^name |supplier02|
        ...
        ^candidate-supplier C6
            ^name |supplier06|
```

Let's Write Some Code!

1. Uncomment the 2nd (“config-priorities”) rule from `agent_starter.soar`.
2. Fill in the rule to match the solution shown below.
3. Run the agent again for 1 step and see what happens!
4. Proceed to the next slide when you're done.

```
sp {elaborate*suppliersort-main*input*config-priorities
    "Print any input priority weights"
    (state <s> ^io.input-link.priorities.<attr> <weight>)
    -->
    (write |INPUT WEIGHT: | <attr> |: | <weight> (crLf))}
```

Final Results

Both your rules together should now print the full set of supplier names and priority weights as shown on the right.

If so, **congratulations!**
You have completed Project 02!

```
**
**
Total: 4 productions sourced.
step
INPUT: Supplier candidate: supplier06
INPUT: Supplier candidate: supplier05
INPUT: Supplier candidate: supplier04
INPUT: Supplier candidate: supplier03
INPUT: Supplier candidate: supplier02
INPUT: Supplier candidate: supplier01
INPUT WEIGHT: speed: 7
INPUT WEIGHT: packaging: 8
INPUT WEIGHT: availability: 8
INPUT WEIGHT: quality: 11
INPUT WEIGHT: sustainability: 11
INPUT WEIGHT: total-cost: 11.010000
      1: ==>S: S2 (state no-change)
```