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

Project 01: Hello World!

By Dr. Bryan Stearns, 2024

Lesson 01 – Outline

This lesson explains the following new concepts:

1. Basic Soar Computation
   - Soar Productions/Rules
   - Soar Working Memory (WM)
2. Writing Soar Rules
   - The Soar CLI
   - Conditions
     - The default "`^type`" Working Memory Element (WME)
   - Actions
     - The `(write)` and `(crlf)` RHS functions

Basic Soar Computation

What your code does

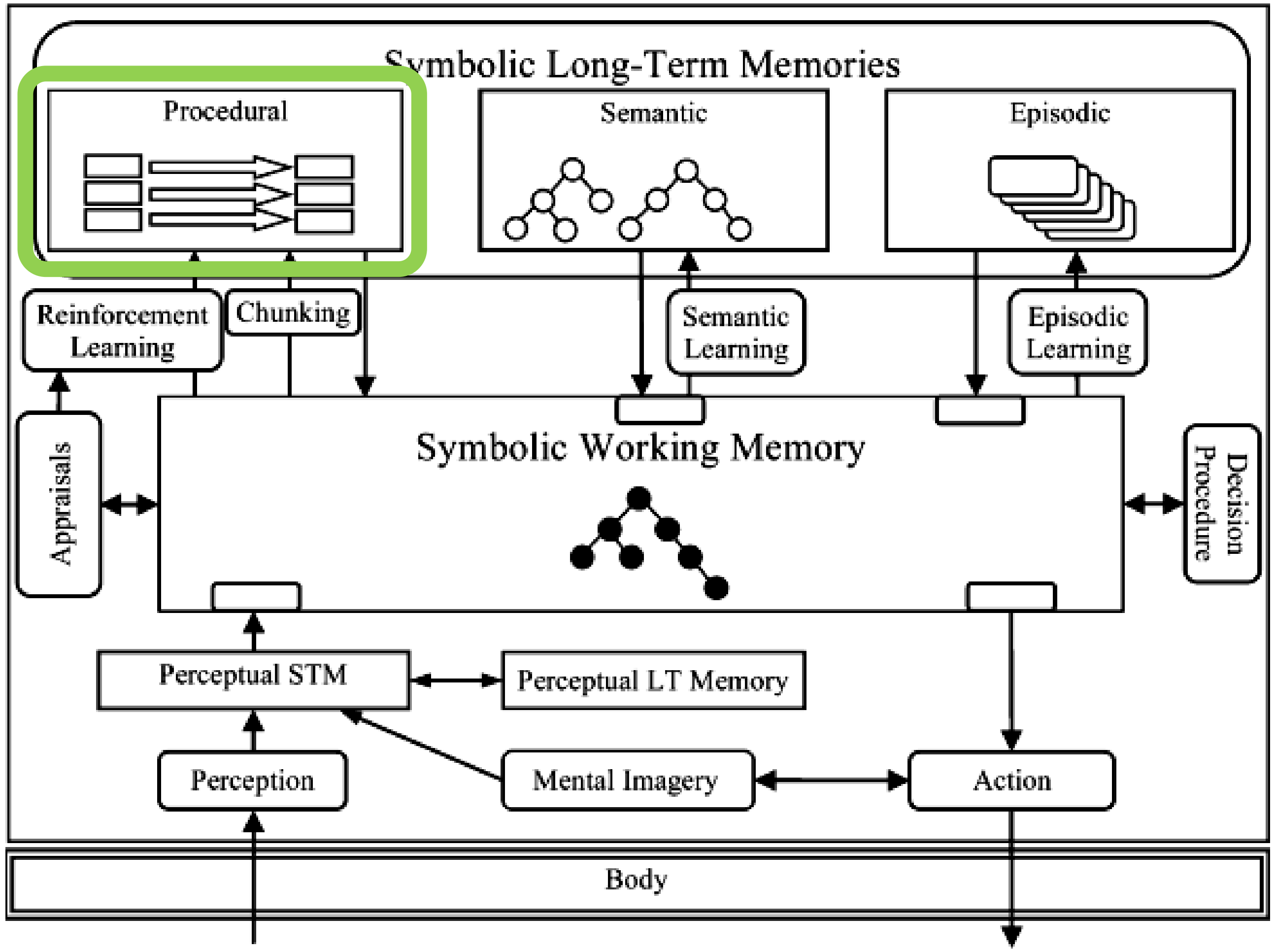
Programming Soar

Program a Soar agent by defining its initial *Procedural Memory*

- Procedural Memory holds **production rules** (or just "rules")

Soar rules:

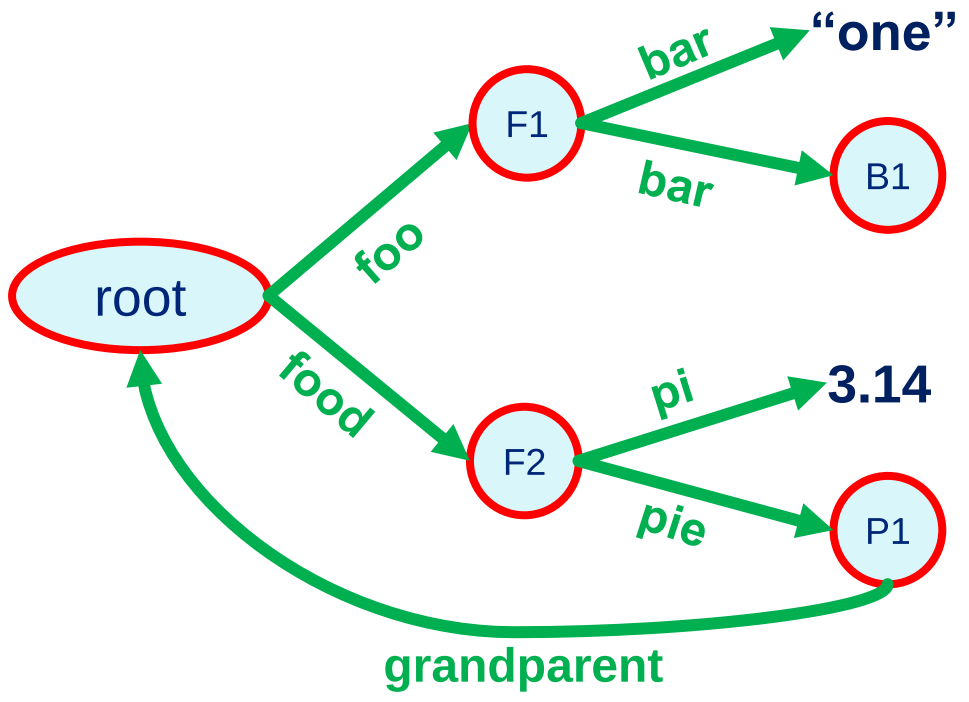
- Are IF -> THEN statements
- Test the Working Memory (WM) graph
- Modify WM when their conditions are satisfied.



Soar Working Memory

WM is a graph:

- Directed
- Cyclic
- Has a single root node



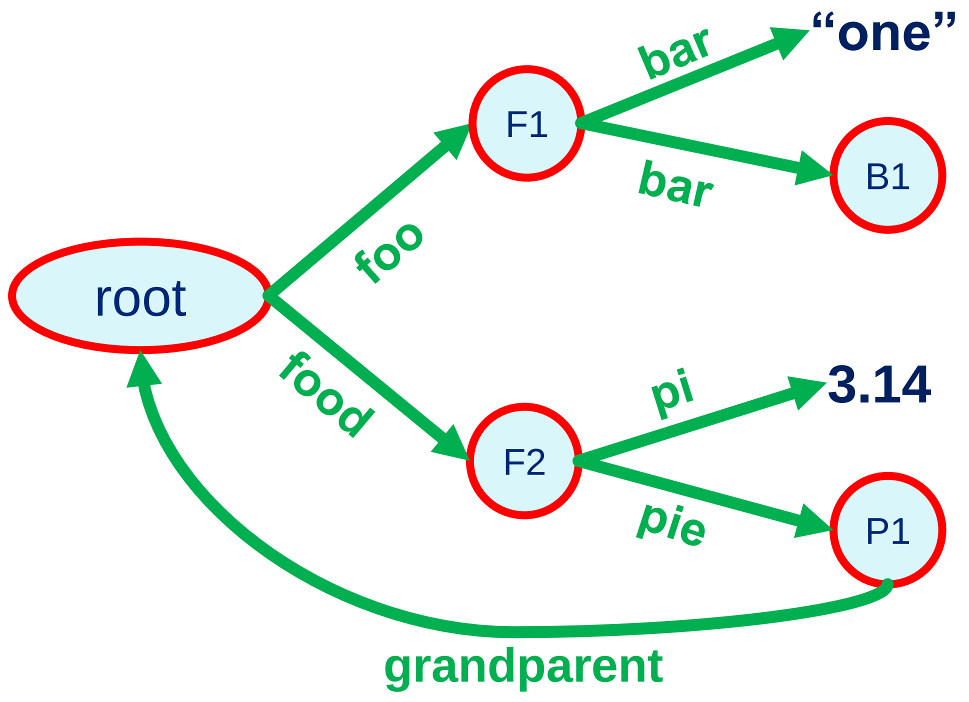
Soar Working Memory: Graph Edges

Edges are always labeled

- Labels do **not** have to be unique

Edges can point to:

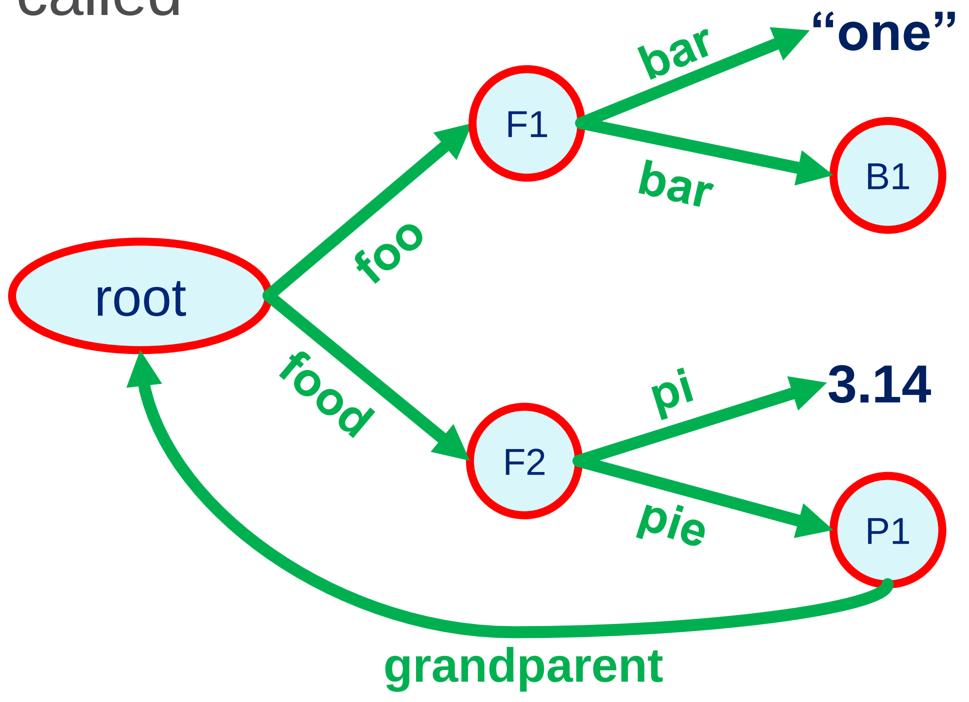
- Nodes
- Strings
- Integers
- Floats



Soar Working Memory Elements

WM is represented as a set of edge tuples called “**Working Memory Elements**” (WMEs).

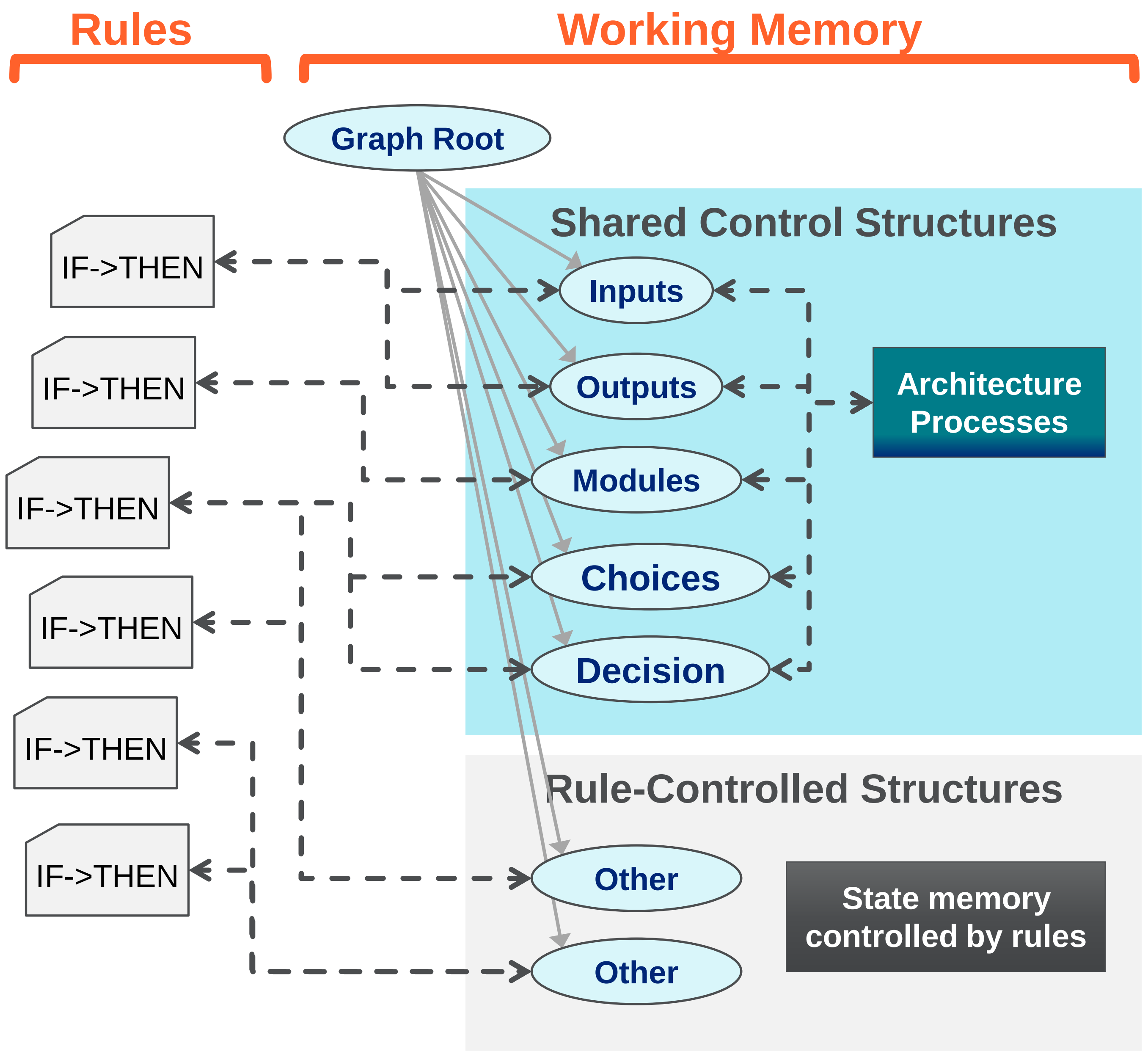
ID	Attribute	Value
root	foo	F1
root	food	F2
F1	bar	“one”
F1	bar	B1
F2	pi	3.14
F2	pie	P1
P1	grandparent	root



“WME” is pronounced “wim-ee”

How Do Soar Rules Work?

- Rules condition on the WM graph and modify it in logical parallel
- Some WM structures are controlled by the architecture
  - Rules interact with automatic architecture processes by reading/modifying these structures
- Most WM structures will be created and controlled by programmed rules



Writing Soar Rules

Coding your first Soar production

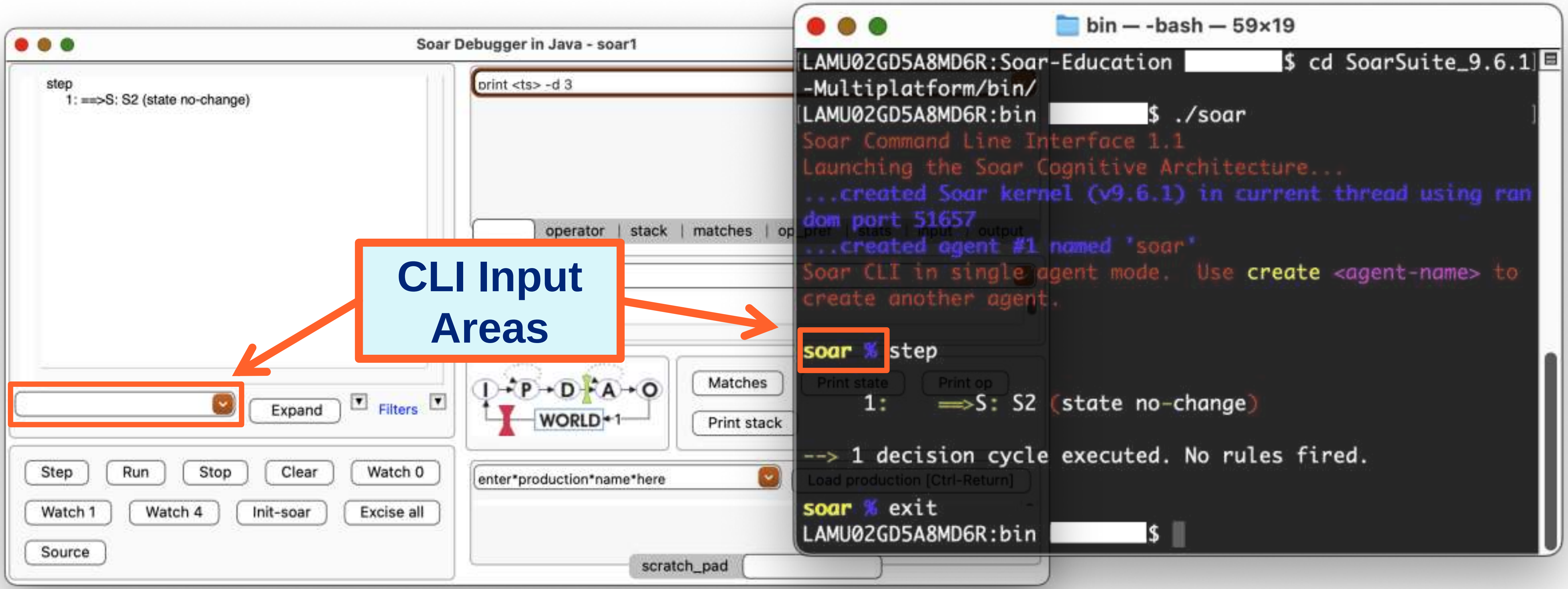
Soar Scripting

Almost all interactions with Soar invoke Soar's Command Line Input (CLI) commands

- Typed into a Soar kernel terminal
- Or loaded from a text file (conventionally given the ".soar" extension).

We'll be using files

CLI Input Areas



```
step
    1: ==>S: S2 (state no-change)
```

```
print <ts> -d 3
```

```
enter*production*name*here
```

```
LAMU02GD5A8MD6R:Soar-Education [redacted]$ cd SoarSuite_9.6.1
-Multiplatform/bin/
LAMU02GD5A8MD6R:bin [redacted]$ ./soar
Soar Command Line Interface 1.1
Launching the Soar Cognitive Architecture...
...created Soar kernel (v9.6.1) in current thread using ran
dom port 51657
...created agent #1 named 'soar'
Soar CLI in single agent mode.  Use create <agent-name> to
create another agent.

soar % step
     1:     ==>S: S2 (state no-change)

--> 1 decision cycle executed. No rules fired.

soar % exit
LAMU02GD5A8MD6R:bin [redacted]$
```

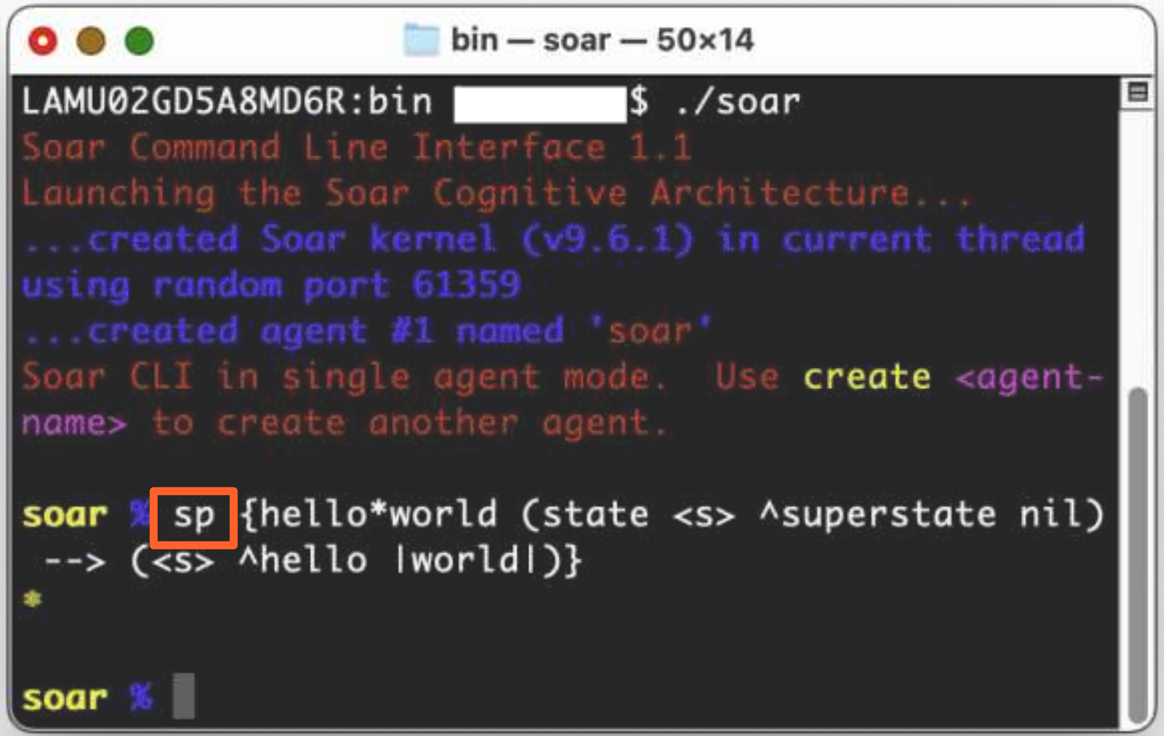
Define a Soar Rule

`sp { … }`

"sp" stands for "Soar Production"

The command that defines a Soar rule is "**sp**"

- *Followed by a space*
- Followed by curly braces that contain the rule definition



```
LAMU02GD5A8MD6R:bin [redacted]$ ./soar
Soar Command Line Interface 1.1
Launching the Soar Cognitive Architecture...
...created Soar kernel (v9.6.1) in current thread
using random port 61359
...created agent #1 named 'soar'
Soar CLI in single agent mode.  Use create <agent-
name> to create another agent.

soar % sp {hello*world (state <s> ^superstate nil)
 --> (<s> ^hello |world|)}
*

soar %
```

Soar Production Syntax

`sp` command syntax:

1. **Rule name**
   - An existing rule with the same name gets overwritten
2. Documentation string (optional)
   - But always a good idea!
3. Rule type (optional)
   - *Ignore this for now*
4. **Conditions**
   - One or more parentheses blocks
5. **A "-->" separator**
6. **Actions**
   - One or more parentheses blocks

No spaces in the rule name

No quotes/apostrophes in the doc string

```
sp {production-name
   "Documentation string”
   :type
   (CONDITIONS)
   -->
   (ACTIONS) }
```

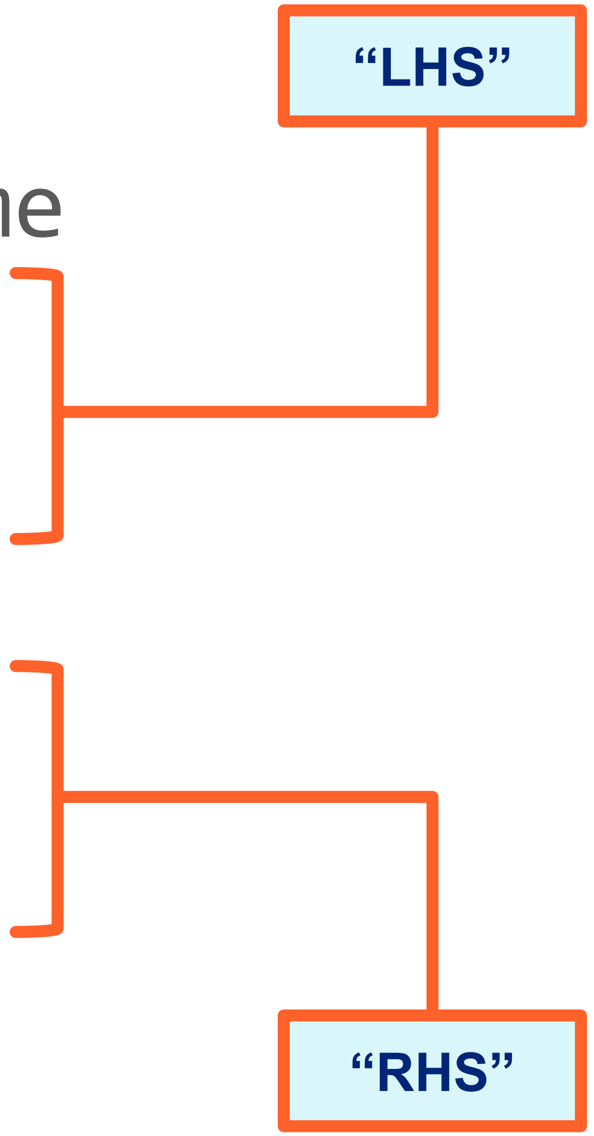
Indentation / line breaks don’t matter

LHS / RHS

The conditions are collectively referred to as the **Left-Hand Side** (LHS)

The actions are collectively referred to as the **Right-Hand Side** (RHS)

```
sp {production-name
    (CONDITION1)
    (CONDITION2)
    -->
    (ACTION1)
    (ACTION2) }
```



Order of Conditions/Actions

The (blocks) of individual conditions or actions within each side can come in any order.

- The conditions are evaluated as a whole.
- The actions are executed in logical parallel.

```
sp {production-name
    (CONDITION1)
    (CONDITION2)
    -->
    (ACTION1)
    (ACTION2) }
```

Order doesn’t matter

Order doesn’t matter

Let’s Write Some Code!

1. Open your `agent_starter.soar` file for Project01.
2. Write the start of your first rule to match what is shown below
   - We’ll replace the underscores with proper content later on
3. Proceed to the next slide when you’re done.

```
sp {hello*world
    (_____)
    -->
    (_____)}
```

Example Rule

This is an example of a complete rule.

Let’s walk through the elements one at a time....

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

LHS Pattern Matching

Rule conditions describe a WM pattern

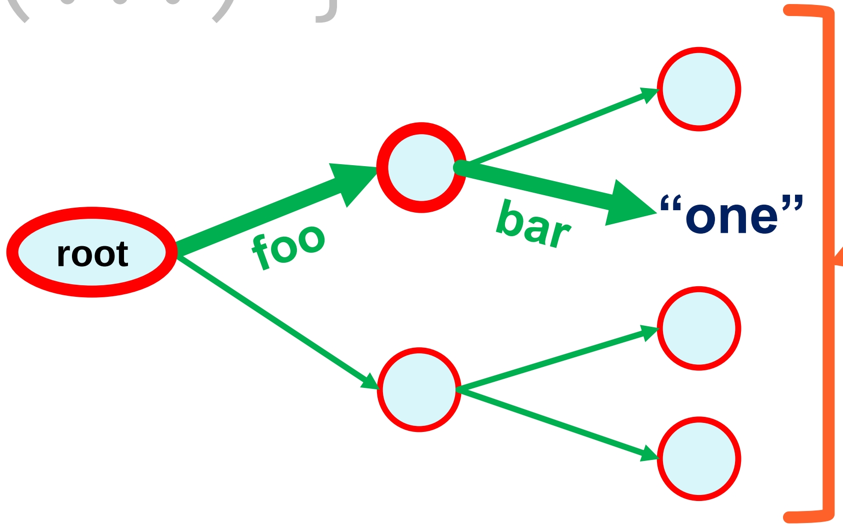
- In terms of WMEs
- All described WMEs must be present for the rule to be satisfied

ID

attribute

value

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



The above rule tests for this pattern.

Rule Syntax: "state"

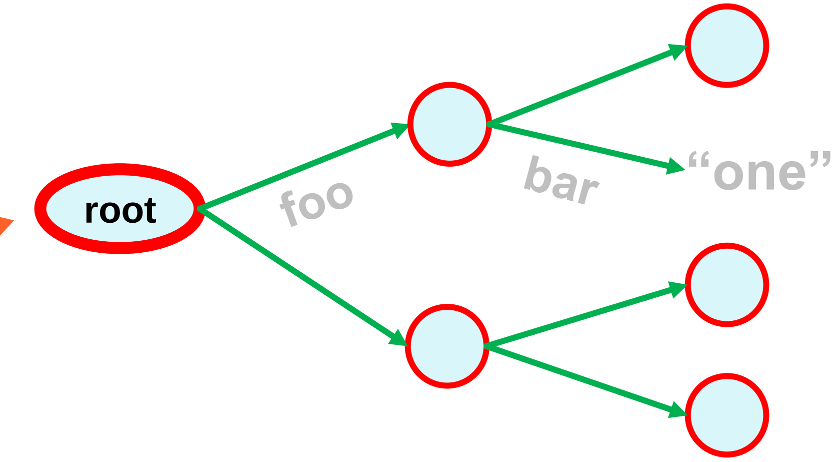
(There can be more than one "state" node in WM, but for now, assume it is the root.)

The pattern must begin at a **state** node

- Use "`state`" in front of a WME ID to mark the ID as the start point

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

The root of the WM graph is a "state" node



Rule Syntax: Variables

Variables:

- Can be used for **ID**, **Attribute**, and/or **Value**
- Are any tokens inside `<angle-brackets>`
  - (No spaces in the name)
- Bind to whatever value in the graph matches the pattern
  - Similar to variables in a query language
- Are only in scope within the containing rule

Binds to the root

Binds to any node that matches `(root ^foo …)`

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

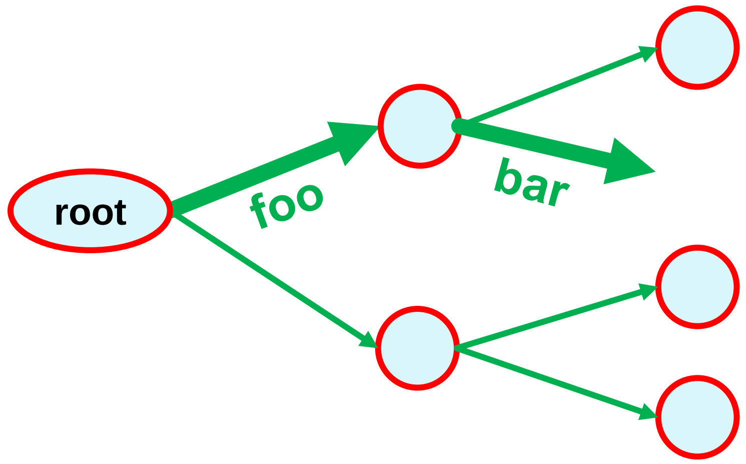
References the same node bound on the first line because it references the same variable name.

Rule Syntax: Attributes

Attributes:

- Must be preceded by the carat ("^") character
- Describe an edge label in the WM graph

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



Rule Syntax: Strings

Soar rules use the pipe (“|”) character to mark strings

The pipe characters are **optional** unless the string has spaces or special characters.

- Soar will automatically infer the data type when it can
- It will read numbers as int or float if they do not have pipes around them

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

This line is equivalent to:
`(<n> ^bar one)`
Soar will infer a string type since there are no spaces or special characters.

Initial WMEs

The Soar architecture creates and maintains several WMEs.

The following are all initial attributes that branch off from the root node:

• **`^io`**	Points to the environment input and output graph structures
• **`^smem`**	Points to the Semantic Memory interface graph structures
• **`^epmem`**	Points to the Episodic Memory interface graph structures
• **`^reward-link`**	Points to the Reinforcement Learning interface graph structure
• **`^superstate`**	Points to the parent state root node, or `|nil|` if there is none
• **`^type`**	The type of the parent node (value will be `|state|`)

We'll learn how to use most of these later.

Initial WMEs

The Soar architecture creates and maintains several WMEs.

The following are all initial attributes that branch off from the root node:

• `^io`	Points to the environment input and output graph structures
• `^smem`	Points to the Semantic Memory interface graph structures
• `^epmem`	Points to the Episodic Memory interface graph structures
• `^reward-link`	Points to the Reinforcement Learning interface graph structure
• `^superstate`	Points to the parent state root node, or `|nil|` if there is none
• **`^type`**	The type of the parent node (value will be `|state|`)

We’ll learn how to use most of these later.

For now, we’ll just focus on the **`type`** WME.

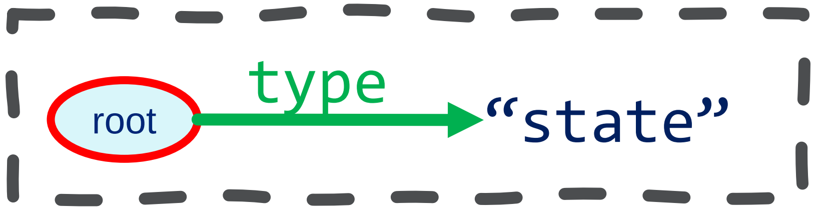
The type WME

The root will always have the type “`state`”.

You can use this to make a rule that always matches.

```
sp {example*rule*2
    (state <s> ^type state)
    -->
    (...) }
```

This condition will always be true.



Let’s Write Some Code!

1. Open your `agent_starter.soar` file again.
2. Modify it using what you’ve learned to match the code below.
3. Proceed to the next slide when you’re done.

```
sp {hello*world
    (state <s> ^type state)
    -->
    (_____)}
```

Rule Actions

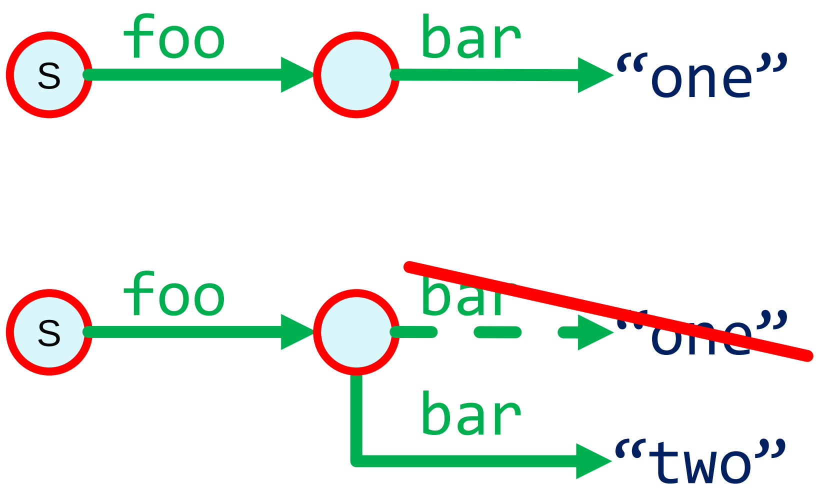
Conditions *test* for the existence of the described pattern.

Actions *create or remove* WMEs from part of that pattern.

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

Adding and Removing WMEs

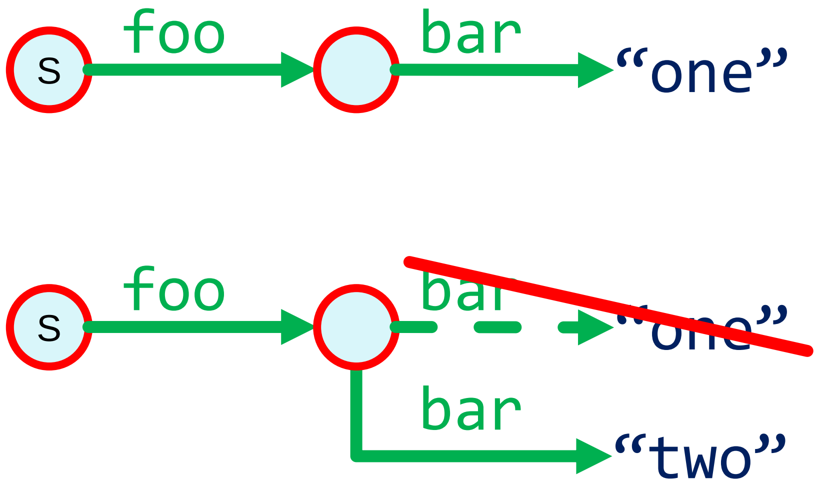
If this graph pattern exists…



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

Replace "one" with "two"
(by removing the "one" WME
and adding the "two" WME)

Adding and Removing WMEs



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

Every ID variable on the RHS must have been bound on the LHS.

Notice the minus sign.
This removes the described WME!

RHS Functions

Actions can also call functions!

- These are called **RHS Functions**

RHS Functions use the following syntax:

- `(func-name [args …] )`

Args must be separated by spaces

```
sp {example*rule*3
    (state <s> ^foo <n>)
    (<n> ^bar |one|)
    -->
    (write |A message!|) }
```

The (write) Function

The **(write)** function prints a message to std-out.

- It takes any number of args
- Converts those to string
- Then concatenates them to form the printed message

```
sp {example*rule*3
    (state <s> ^foo <n>)
    (<n> ^bar |one|)
    -->
    (write |A message!|) }
```

A single arg

The `(crlf)` Function

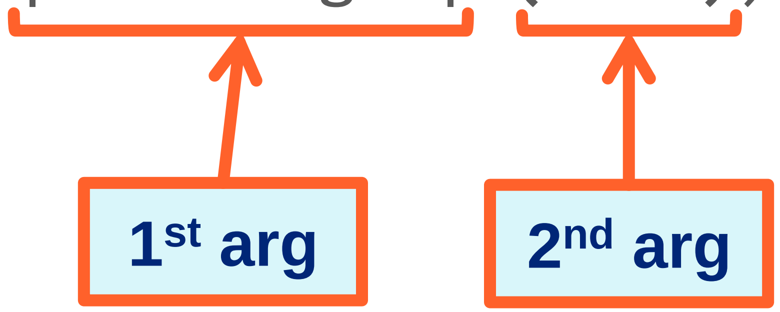
It is useful to combine `(write)` with **`(crlf)`**

- **`(crlf)`** returns a newline character
- It takes no arguments
- CRLF stands for “Carriage Return Line Feed”

You will usually want to end `(write)` messages with a `(crlf)`.

- So later messages start on a new line

```
sp {example*rule*3*with-crlf
    (state <s> ^foo <n>)
    (<n> ^bar |one|)
    -->
    (write |A message!| (crlf)) }
```



Provided RHS Functions

Soar provides many useful RHS Functions

- See the Soar Manual (pg. 71) for documentation

You can also write your own

- See the Soar Manual (pg. 80)

Function	Description
`(interrupt)`	Pauses Soar
`(halt)`	Terminate Soar
`(+), (-), (*), (/)`	Math operations
`(int), (float)`	Convert to the indicated data type
`(min), (max)`	The min and max operations
`(ifeq)`	Conditionally return a value
`(size)`	Get the count of edges from a node

A sampling of some useful RHS Functions

Let’s Write Some Code!

1. Open your `agent_starter.soar` file again.
2. Fill what remains so that the rule matches the code shown below
3. Run it in Soar using the `run_project.py` script.
   - Click the “Step” button once in the SoarJavaDebugger window.
   - You should see “Hello world!” appear in the main display area.
4. **Congratulations!** You have run your first Soar agent!

```
sp {hello*world
    “Your first Soar rule”
    (state <s> ^type state)
    -->
    (write |Hello world!| (crlf))}
```