

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

Project 07: Practice Debugging

By Dr. Bryan Stearns, 2024



Problem

We have a substate set up,

- But our agent doesn't yet use it to sort suppliers with complex reasoning.
- We still sort only by total-score.

We also don't have much experience using debugging commands to inspect our code when things go wrong.

Solution

Use substate operators to:

- Iterate over attribute weights given on the input-link
 - In descending order
- For each weight, sort suppliers by their attribute scores for that weight.

Get practice using the matches -w command for rule debugging!

Project Goal

We will use our substate to sort suppliers based on:

- The score for each attribute for each supplier relative to other suppliers
- The weight of each attribute relative to other attributes

Not just a single attribute like total-score.

Lesson 07 – Outline

This lesson explains the following new concepts:

- Copying down superstate WMEs for efficiency
- Nested condition conjunctions within rules
- Debugging with matches -w
- LHS conjunctions and disjunctions

This lesson starts to put you in the front seat:

- More coding design up to you!
- More debugging up to you!

Lesson Focus

This lesson **will**:

- Equip you with concepts that you will find useful as you develop your code
- Show you debugging techniques to empower you to code more independently

This lesson **will not**:

- Walk you through each step of coding this project

Sorting Logic Review

The Goal for Supplier Sort

Input Review: Attribute Weights and Rank Scores

Our input for Supplier Sort comes in two types of structures:

1. Priorities

- A single object that gives the global priority weight for each attribute type
 - These weights apply for the whole task across all suppliers
- Each supplier is expected to have a value for each of these attributes
- Example: Attribute “quality” has a higher weight than attribute “speed”.

2. Candidate Supplier

- Gives the name and other attribute scores for a particular supplier.
 - For total-cost, lower is better
 - For all other weighted attributes, higher is better
- There is a separate `^candidate-supplier` object per supplier.
 - Example: Supplier01 has “quality” score of 1 and a “speed” score of 2.
- There are two special attributes:
 - total-score: The sum of all rank scores (non-cost)
 - total-sats: The count of data needs this supplier can satisfy

```
S1 ^io.input-link I2
  ^priorities P1
    ^total-cost 11.01
    ^sustainability 11
    ^quality 10
    ^availability 8
    ^packaging 8
    ^speed 7

  ^candidate-supplier C1
    ^name |supplier01|
    ^total-score 12
    ^total-sats 2
    ^total-cost 35.0
    ^sustainability 3
    ^availability 3
    ^quality 1
    ^packaging 3
    ^speed 2
    ...
```

Sorting Steps

By the end of this tutorial series, we want to have the following routine:

1. Exclude suppliers where total-sats == 0
2. Sort suppliers by total-sats in descending order
- 3. **Iteratively sort suppliers by attributes in priority weight order**
- 4. Sort remaining tied suppliers by total-score
5. Sort remaining tied suppliers randomly

We've already implemented step 4.

**We will now implement step 3.
(described further in the next slide)**

```
S1 ^io.input-link I2
    ^priorities P1
        ^total-cost 11.01
        ^sustainability 11
        ^quality 10
        ^availability 8
        ^packaging 8
        ^speed 7

    ^candidate-supplier C1
        ^name |supplier01|
        ^total-score 12
        ^total-sats 2
        ^total-cost 35.0
        ^sustainability 3
        ^availability 3
        ^quality 1
        ^packaging 3
        ^speed 2

    ...
```


Iterating Through Weights

Our agent should sort suppliers by their scores in priority weight order.

Example:

- total-cost has the highest weight at 11.01, so first sort suppliers in ascending order of their total-cost score:

total-cost	25.0	25.0	25.0		35.0	35.0	35.0
operator's supplier	supplier03	supplier04	supplier05	>	supplier01	supplier02	supplier06

- completeness has the next-highest weight at 11, so sort remaining suppliers in descending order by their completeness score:

completeness	3	3	2		3	2	2
operator's supplier	supplier04	supplier05	supplier03		supplier01	supplier02	supplier06

- and so on...

```
S1 ^io.input-link I2
  ^priorities P1
    ^total-cost 11.01
    ^sustainability 11
    ^quality 10
    ^availability 8
    ^packaging 8
    ^speed 7

    ^candidate-supplier C1
      ^name |supplier01|
      ^total-score 12
      ^total-sats 2
      ^total-cost 35.0
      ^sustainability 3
      ^availability 3
      ^quality 1
      ^packaging 3
      ^speed 2
      ...
```

Handling Tied Weights

What if two attributes share the same weight?

- Then sort by the *sum* of those attribute scores

Example: Say completeness and quality **both have weight 11**

completeness + quality	3+3	3+3		3+2		2+3		3+1		2+1
operator's supplier	supplier03	supplier05	>	supplier04		supplier02	>	supplier01	>	supplier03

For now, though, we will pretend that weights are always unique.

We will account for tied weights in Project 08.

```
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|
    ^total-score 12
    ^total-sats 2
    ^total-cost 35.0
    ^sustainability 3
    ^availability 3
    ^quality 1
    ^packaging 3
    ^speed 2

  ...
```

STEP 1: Copy Superstate WMEs

Prepping Our Substate For Easy Use

Starting Point: Define Your WMEs

When working in a substate, *plan ahead* what superstate WMEs you will use often.

- Then copy them to the substate for easy access in the rules you write, like so:

```
sp {elaborate*my-tie*superstate*fizz-buzz
    (state <s> ^superstate.io.input-link.fizz <buzz>)
    -->
    (<s> ^fizz <buzz>)}
```

1. This simplifies the rest of your code.

- It is easier to reference (<s> ^fizz <buzz>)
than (<s> ^superstate.io.input-link.fizz <buzz>)

2. This also can make backtracing simpler:

- It provides backtracing with a single point of reference to the superstate WME, rather than one for each rule that separately accesses the same superstate WME.

STEP 1: Copy Input Priority WME

1. Open your `agent_starter.soar` file and find the section labeled “STEP 1”.
2. Write a rule there to copy the input-link `^priorities` WME to the substate.
 - Name the rule “`elaborate*suppliersort-tie*priorities`”
3. Test on your own whether your rule works:
 - After your agent has had a chance to fire rules in the substate (step 3), ensure that S2 has a WME that points to the input-link priorities ID.
 - Make sure it’s the same ID that is used on the input-link, such as “P1”.

Commands you might use:

- “step” (step 1 decision cycle)
- “e” (step 1 elaboration cycle)
- “print S2” (print the WMEs under S2)
- “print I2” (print the WMEs under the input-link, which always has this ID)

Rule Naming Conventions

Knowing Your Rule at a Glance

Rule Naming Conventions

When naming your rules, the following convention is suggested:

Separate the following elements by asterisks:

1. The rule type (“elaborate” / “propose” / “prefer” / “apply”)
2. The state name it is supposed to fire in (such as “suppliersort-main”)
3. The operator name (if any is relevant)
4. Any additional key words as needed that indicate what makes this rule unique from others that share the same naming for elements 1-3

1

2

3

4

```
sp {propose*suppliersort-tie*my-operator-name*special-case-42
```

There are other conventions you can use (such as swapping the order of steps 1 and 2)

- *But be consistent!*
- A good naming scheme gives you confidence that your rule names are unique.
- It also helps you know immediately what your rule does just from the name.

Debugging Rule Matching

How To Diagnose a LHS Bug

The matches -w [rule_name] Command

The first thing to check when your rule doesn't seem to fire when you thought it would:

- `matches -w your*rule*name*here`

This will show which parts of your rule's LHS are *currently satisfied*.

Dot notation is expanded

The leading number before each condition indicates how many WMEs match that condition.
(Here, there are 2 states, but only 1 of them has ^io, and that ^io structure only has 1 ^input-link.)

If you see ">>>>" before a condition, it means that condition was *not found* in WM.
(Here, this helps me see that "priorities" is misspelled, which is why the rule doesn't match.)

The "Matches For Left/Right" show what WMEs Soar was able to match on until it hit the non-matching condition.

```
2 (state <s> ^superstate <s*1>)
1 (<s*1> ^io <i*1>)
1 (<i*1> ^input-link <i*2>)
>>>> (<i*2> ^priorities <p>)
*** Matches For Left ***
(114: S2 ^superstate S1)
(11: S1 ^io I1)
(12: I1 ^input-link I2)
*** Matches for Right ***

0 complete matches.
```

The matches -w [rule_name] Command

The first thing to check when your rule doesn't seem to fire when you thought it would:

- `matches -w your*rule*name*here`

This will show which parts of your rule's LHS are *currently satisfied*.

After fixing the typo, I see at least a “1” for each condition.

The command output then shows me which
WM IDs matched my rule.
(This can be very useful for diagnosing my rule
if it fired when I *didn't* expect it to.)

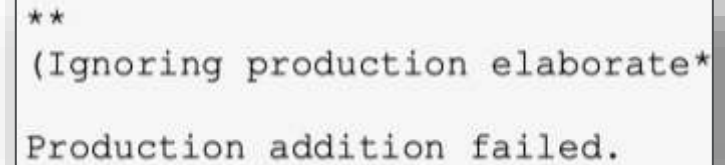
```
2 (state <s> ^superstate <s*1>)
1 (<s*1> ^io <i*1>)
1 (<i*1> ^input-link <i*2>)
1 (<i*2> ^priorities <p>)

1 complete matches.
*** Complete Matches ***
(114: S2 ^superstate S1)
(11: S1 ^io I1)
(12: I1 ^input-link I2)
(80: I2 ^priorities P1)
```

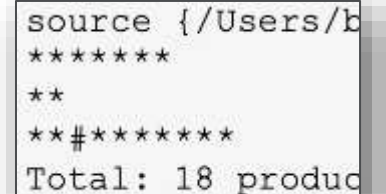
Other Steps When Debugging A Rule

“My rule isn’t working. Why doesn’t my rule work?”

1. Check that you don’t have syntax errors in your rules.
 - Soar will print error/warning messages if it detects any.
 - (And will then abort loading further rules)
 - Common errors:
 - A starting variable in your RHS doesn’t match anything on the LHS
 - Forgetting a space in “sp {“
 - Forgetting to mark your root <s> ID with “state”.
2. Check that your rule doesn’t share a name with other rules.
 - If two rules share the same name, the one imported second will *overwrite the first*.
 - If you see a “#” among the “*”s when importing rules, it means a rule got overwritten.
3. Check that you are inspecting a point in time when your rule is *supposed* to have fired.
 - For instance: An apply rule won’t fire until the Apply Phase.
4. Check that it didn’t fire sooner than you expected.
 - If the matches command says your rule matched completely, but you don’t see its effects in WM, it might have fired a while ago, and you might have other rules that already removed its effects.



```
**
(Ignoring production elaborate*
Production addition failed.
```



```
source {/Users/b
*****
**
**#*****
Total: 18 produc
```

STEP 2: Propose (evaluate-weight)

An Operator to Iterate Over Weights

STEP 2: Propose (evaluate-weight)

Find the comment section in `agent_starter.soar` labeled “STEP 2”

- Follow the instructions there to propose an (evaluate-weight) operator with indifferent (“=”) preference.
- It should be proposed for each input priority weight.

After you make your rule, test it by running the agent for 3 steps.

```
step 3
  1: 0: 01 (init)
  2: ==>S: S2 (operator tie)
  3:      ==>S: S3 (operator tie)
```

You should get an operator tie (S2) for tied (select-supplier) operators and then another operator tie (S3) for a tie involving (evaluate-weight). Run the `matches -w` command on your rule. You should see 6 matches →

```
matches -w propose*suppliersort-tie*evaluate-weight

1 (state <s> ^name suppliersort-tie)
1 (<s> ^priorities <p*1>)
6 (<p*1> ^<attr> <weight>)
6 (<s> -^weight-evaluated <weight>)

6 complete matches.
*** Complete Matches ***
(137: S2 ^name suppliersort-tie)
(138: S2 ^priorities P1)
(77: P1 ^total-cost 11.010000)

(137: S2 ^name suppliersort-tie)
(138: S2 ^priorities P1)
(78: P1 ^sustainability 11)

(137: S2 ^name suppliersort-tie)
(138: S2 ^priorities P1)
(79: P1 ^quality 11)

(137: S2 ^name suppliersort-tie)
(138: S2 ^priorities P1)
(80: P1 ^availability 8)

(137: S2 ^name suppliersort-tie)
(138: S2 ^priorities P1)
(81: P1 ^packaging 8)

(137: S2 ^name suppliersort-tie)
(138: S2 ^priorities P1)
(82: P1 ^speed 7)
```

<	state	operator	stack	matches	op_pref	stats	input	output
---	-------	----------	-------	---------	---------	-------	-------	--------

STEP 2: Diagnose the Tie

Why does (evaluate-weight) lead to a tie when it has indifferent preference?

Figure it out: Use the “preferences <ss> operator --names” command to see competing preferences.

- The command is provided in a tab in the Java Debugger.
- (You might have to replace the <ss> component of the command though.)

You should see output like what is shown to the right. →

What does this tell us?

- The tie is between (evaluate-weight) and (break-attr-tie).
- Not between different (evaluate-weight) operators.

We need our agent to prefer (evaluate-weight) over (break-attr-tie).

```
preferences <ss> operator --names
Preferences for S2 ^operator:

acceptables:
  010 (evaluate-weight) + :I [level 2]
    From propose*suppliersort-tie*evaluate
  011 (evaluate-weight) + :I [level 2]
    From propose*suppliersort-tie*evaluate
  012 (evaluate-weight) + :I [level 2]
    From propose*suppliersort-tie*evaluate
  013 (evaluate-weight) + :I [level 2]
    From propose*suppliersort-tie*evaluate
  014 (evaluate-weight) + :I [level 2]
    From propose*suppliersort-tie*evaluate
  015 (evaluate-weight) + :I [level 2]
    From propose*suppliersort-tie*evaluate
  09 (break-attr-tie) + :I [level 2]
    From propose*suppliersort-tie*break-

unary indifferents:
  010 (evaluate-weight) = :I [level 2]
    From propose*suppliersort-tie*evaluate
  011 (evaluate-weight) = :I [level 2]
    From propose*suppliersort-tie*evaluate
  <
state operator stack matche op_pref tats input
```

STEP 3: (break-attr-tie) Is the Worst

Perform (evaluate-weight) Before It

STEP 3: Modify (break-attr-tie)'s Preference

Find the comment section in `agent_starter.soar` labeled “STEP 3”

- Follow the instructions so (break-attr-tie) is proposed with worst (“<”) preference.

After making this small change, run the agent from the beginning for 3 steps.

- You should now see the following:

```
step 3
  1: 0: 01 (init)
  2: ==>S: S2 (operator tie)
  3:    0: 010 (evaluate-weight)
```

Proceed to the next slide when you're ready for STEP 4.

STEP 4:

Evaluate Weights in Order

In Descending Order of Weight Value

STEP 4: Evaluate Weights in Descending Order

Find the comment section in `agent_starter.soar` labeled “STEP 4”

- Follow the instructions: Make a preference rule for (evaluate-weight).
- Your rule should prefer one evaluate-weight operator over another if it has a higher weight than the other.
 - (Check that your proposal rule from STEP 2 attached this weight to the operator when it proposed the operator, so that you can reference it here in this preference rule.)

After adding this rule, run the agent from the beginning for 3 steps.

- The `matches -w` command should show you something like this →
- Proceed to the next slide when you are done.

```
matches -w prefer*suppliersort-tie*evaluate-weight*higher-weight
1 (state <s> ^name suppliersort-tie)
7 (<s> ^operator <o2> +)
6 (<o2> ^name evaluate-weight)
42 (<s> ^operator <o1> +)
36 (<o1> ^name evaluate-weight)
36 (<o2> ^weight <w*1>)
13 (<o1> ^weight { > <w*1> <v1> })

13 complete matches.
*** Complete Matches ***
(137: S2 ^name suppliersort-tie)
(163: S2 ^operator 014 +)
(153: 014 ^name evaluate-weight)
(164: S2 ^operator 015 +)
(156: 015 ^name evaluate-weight)
(155: 014 ^weight 11)
(158: 015 ^weight 11.010000)

(137: S2 ^name suppliersort-tie)
(162: S2 ^operator 013 +)
(150: 013 ^name evaluate-weight)
```

state	operator	stack	matches	op_pref	stats	input	c
-------	----------	-------	---------	---------	-------	-------	---

STEP 5: Applying (evaluate-weight)

The First Apply Rule: total-cost

(evaluate-weight) Apply Rules Objective

Our next step will be to apply our (evaluate-weight) operator.

We will need multiple apply rules. The main two rules will look something like this:

1. IF: (evaluate-weight) is selected,
AND for two tied items in this state,
one has a **lower total-cost** value,
AND this operator's attribute is **total-cost**,
THEN:
In the superstate, prefer the item with the **lower cost** over the other item.
2. IF: (evaluate-weight) is selected,
AND for two tied items in this state,
one has a **higher score** for this (evaluate-weight) operator's attribute,
AND that attribute is **not total-cost**,
THEN:
In the superstate, prefer the item with the **higher score** over the other item.

STEP 5: Sort by total-cost

Find “STEP 5” in `agent_starter.soar` and uncomment the given rule outline there.

- Follow the instructions to make the first apply rule (this will be a complex rule!):
- NOTE: You will probably want to avoid using dot-notation for this rule.
You will need to use each ID.

The supplier score is located here:
`<s> ^item.supplier.<attr> <score>`

IF: (evaluate-weight) is selected,
 AND for two tied items in this state,
 one has a **lower total-cost** value,
 AND this operator's attribute is **total-cost**,

THEN:

In the superstate, prefer the item with the **lower cost** over the other item.

Return a result to the superstate:
`<ss> ^operator <o1> > <o2>`

STEP 5: Run and Test

To test your apply rule, run your agent for 4 steps.

You should see something like this →

Check for each of the following:

- Your rule prints 9 preference messages
- Suppliers 3, 4, and 5 are preferred over suppliers 1, 2, and 6
- Your rule tests that the selected (evaluate-weight) operator's attribute is total-cost.
- (The rule will look like it still works without this test, because the agent only selects total-cost's weight so far anyway. But if the agent selected another weight, the logic would fail.)

```
step 4
  1: 0: 01 (init)
  2: ==>S: S2 (operator tie)
  3:    0: 015 (evaluate-weight)
total-cost: Prefer supplier03 > supplier01
total-cost: Prefer supplier04 > supplier01
total-cost: Prefer supplier05 > supplier01
total-cost: Prefer supplier03 > supplier02
total-cost: Prefer supplier04 > supplier02
total-cost: Prefer supplier05 > supplier02
total-cost: Prefer supplier03 > supplier06
total-cost: Prefer supplier04 > supplier06
total-cost: Prefer supplier05 > supplier06
  4:    ==>S: S3 (operator no-change)
```

**If your rule matches on incorrect supplier pairs, use the
matches -w command to debug it!**

STEP 6: Completing a Weight

So the (evaluate-weight) Proposal Retracts

STEP 6: Marking a Weight as Evaluated

In order to avoid op no-change impasses, we need the (evaluate-weight) operator's proposal to retract once the weight is evaluated.

Look back to the comment instructions for the proposal rule (STEP 2).

- See that the proposal should test for the absence of a ^weight-evaluated WME.

We will write an apply rule that will create this WME for the selected operator's weight.

Uncomment the “STEP 6” rule outline and fill in the underscores according to the instructions.

- The LHS only needs to reference the current operator's ^name, ^weight, and ^attribute WMEs.
- (The ^attribute WME is just for the sake of the (write) message, which is useful for seeing what the agent is doing.)

STEP 6: Run and Test

To test your rule, you can now run your agent from start to finish:
After sorting by `total-cost`, your agent should iterate through each weight in descending order.

- It won't do any sorting for non-cost weights *yet*.
- After exhausting all weights, it will then sort by `total-score` using (`break-attr-tie`).

If instead you still get an op no-change impasse:

- Check that your proposal rule tests for the same WME as you create in your new apply rule (tests that it is missing).
- Check that your new apply rule creates the correct value for `^weight-evaluated` (no typos in your variable names).
- Inspect your proposal rule with `matches -w` after your apply rule fires to see if/why it is still matching.

```
total-cost: Prefer supplier05 > supplier02
total-cost: Prefer supplier03 > supplier06
total-cost: Prefer supplier04 > supplier06
total-cost: Prefer supplier05 > supplier06
  * Evaluated weight 11.010000 (attribute "total-cost")
    4: 0: 013 (evaluate-weight)
  * Evaluated weight 11 (attribute "quality")
    5: 0: 011 (evaluate-weight)
  * Evaluated weight 8 (attribute "packaging")
    6: 0: 010 (evaluate-weight)
  * Evaluated weight 7 (attribute "speed")
    7: 0: 09 (break-attr-tie)
Total-score: Prefer supplier05 > supplier03
Total-score: Prefer supplier03 > supplier04
Total-score: Prefer supplier05 > supplier04
  8: 0: 06 (select-supplier)
** OUTPUT: First try supplier05
  9: 0: 04 (select-supplier)
** OUTPUT: Then try supplier03
 10: 0: 05 (select-supplier)
** OUTPUT: Then try supplier04
 11: ==>S: S3 (operator tie)
 12: 0: 022 (evaluate-weight)
  * Evaluated weight 11.010000 (attribute "total-cost")
 13: 0: 021 (evaluate-weight)
  * Evaluated weight 11 (attribute "sustainability")
 14: 0: 018 (evaluate-weight)
  * Evaluated weight 8 (attribute "packaging")
 15: 0: 017 (evaluate-weight)
  * Evaluated weight 7 (attribute "speed")
 16: 0: 016 (break-attr-tie)
Total-score: Prefer supplier01 > supplier02
Total-score: Prefer supplier06 > supplier02
Total-score: Prefer supplier01 > supplier06
 17: 0: 02 (select-supplier)
** OUTPUT: Then try supplier01
 18: 0: 07 (select-supplier)
** OUTPUT: Then try supplier06
 19: 0: 03 (select-supplier)
** OUTPUT: Then try supplier02
 20: 0: 08 (output-supplier-list)
*** DONE ***
```

Condition Conjunctions and Disjunctions

Setting Multiple Constraints Per LHS Value

How To Combine Conditions?

Our remaining apply rule should sort for attributes that are **not** |total-cost|. How do we test that?

- (`<o> ^attribute <> |total-cost|`)

But we also need to get a supplier's score value for the selected operator's attribute:

- (`<o> ^attribute <attr>`)
(`<sup1> ^<attr> <value>`)


So how do you combine (`<o> ^attribute <> |total-cost|`) and (`<o> ^attribute <attr>`)?

- You might try to include both conditions separately...
 - But this could lead to an error if `<o>` ever had more than one `^attribute` WME.
 - One `^attribute` condition might match on a different WME from the other.

Example:

```
(<o> ^attribute <> |total-cost|)  
(<o> ^attribute <attr>)  
(<sup1> ^<attr> <value>)
```

Could match

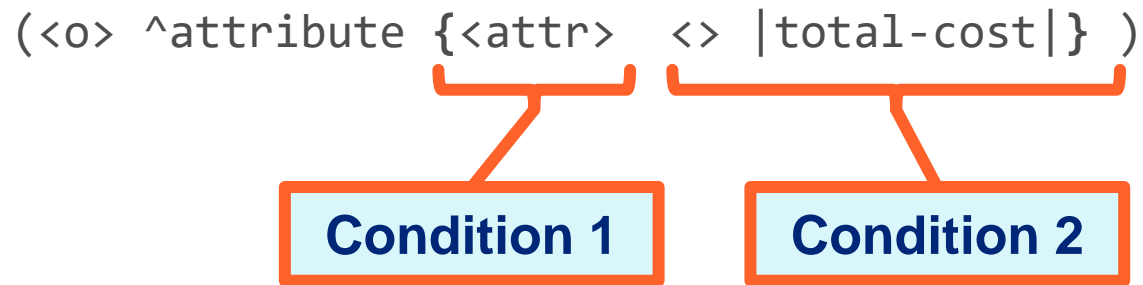
 **x**

```
(01 ^attribute |quality|)  
(01 ^attribute |total-cost|)  
(C2 ^total-cost 25.0)
```

Condition Conjunctions

The solution: **Conjunctions!**

- Soar lets you use curly brackets to combine multiple tests for a single WME:



You can combine as many tests as you like within the curly brackets.

- Order doesn't matter. The above is the same as the following:

`(<o> ^attribute {<> |total-cost| <attr>})`

Condition Disjunctions

If Soar supports Conjunctions (AND logic), does it also support Disjunctions (OR logic)?

- Yes! (You won't need it for this project, but we'll explain it here for completeness' sake.)

To test that a WME satisfies a disjunction of values, use double angle brackets (<< >>).

The following is satisfied if the value is *either* |total-cost| or |total-sats|:

```
(<o> ^attribute << total-cost total-sats >>)
```

Disjunctions **must** have spaces between the angle brackets and the contents.

✗ (<s> ^fizz <<buzz1 buzz2>>)

✓ (<s> ^fizz << buzz1 buzz2 >>)

Disjunctions may **not** include variables nor be used with binary comparisons.

✗ (<s> ^fizz << <var> buzz >>)

✗ (<s> ^fizz < << buzz1 buzz2 >>)

✗ (<s> ^fizz << buzz1 < buzz2 >>)

✓ (<s> ^fizz {<var> << buzz1 buzz2 >>})

STEP 7: Applying (evaluate-weight)

The Last Apply Rule: Non-Costs

STEP 7: Sort by Non-Cost Attributes

Find “STEP 7” in `agent_starter.soar` and uncomment the given rule outline there.

- Follow the instructions to fill in the underscores for this remaining apply rule.
- (You can copy a lot from your solution to STEP 5.)

IF: (evaluate-weight) is selected,
 AND for two tied items in this state,
 one has a **higher score** for this (evaluate-weight) operator’s attribute,
 AND that attribute **is not total-cost**,
THEN:
 In the superstate, prefer the item with the **higher score** over the other item.

Use a conjunction to ensure
<attr> <> |total-cost|,
as per the previous slides

Remember to reverse the logic from the first apply rule:
Higher scores are better.

STEP 7: Run and Test

After finishing STEP 7, you should be able to run and see the following →

The agent sorts tied suppliers by each weight in descending order:

- total-cost: Suppliers 3, 4, 5 > Suppliers 1, 2, 6
 - At this point, only Suppliers 3, 4, and 5 are still tied.
 - So Suppliers 1, 2, and 6 are omitted from the rest of substate S2.
- quality: Suppliers 3, 5 > Supplier 4
- packaging: No sorting (Suppliers 3 and 5 are the same)
- speed: Supplier 5 > Supplier 3

The agent outputs Suppliers 5 > 3 > 4, then hits a new tie for the rest:

- total-cost: No sorting
- quality: Supplier 2 > Suppliers 1, 6

The agent outputs Supplier 2, then hits a new tie for the remainder:

- ... speed: Supplier 1 > Supplier 6

The agent outputs Supplier 1 then 6, and then is done!

S2

```
1: 0: 01 (init)
2: ==>S: S2 (operator tie)
3: 0: 015 (evaluate-weight)
total-cost: Prefer supplier03 > supplier01
total-cost: Prefer supplier04 > supplier01
total-cost: Prefer supplier05 > supplier01
total-cost: Prefer supplier03 > supplier02
total-cost: Prefer supplier04 > supplier02
total-cost: Prefer supplier05 > supplier02
total-cost: Prefer supplier03 > supplier06
total-cost: Prefer supplier04 > supplier06
total-cost: Prefer supplier05 > supplier06
* Evaluated weight 11.010000 (attribute "total-cost")
4: 0: 013 (evaluate-weight)
quality: Prefer supplier03 > supplier04
quality: Prefer supplier05 > supplier04
* Evaluated weight 11 (attribute "quality")
5: 0: 011 (evaluate-weight)
* Evaluated weight 8 (attribute "packaging")
6: 0: 010 (evaluate-weight)
speed: Prefer supplier05 > supplier03
* Evaluated weight 7 (attribute "speed")
7: 0: 06 (select-supplier)
** OUTPUT: First try supplier05
8: 0: 04 (select-supplier)
** OUTPUT: Then try supplier03
9: 0: 05 (select-supplier)
** OUTPUT: Then try supplier04
10: ==>S: S3 (operator tie)
11: 0: 022 (evaluate-weight)
* Evaluated weight 11.010000 (attribute "total-cost")
12: 0: 021 (evaluate-weight)
sustainability: Prefer supplier01 > supplier02
sustainability: Prefer supplier01 > supplier06
* Evaluated weight 11 (attribute "sustainability")
13: 0: 02 (select-supplier)
** OUTPUT: Then try supplier01
14: ==>S: S4 (operator tie)
15: 0: 029 (evaluate-weight)
* Evaluated weight 11.010000 (attribute "total-cost")
16: 0: 028 (evaluate-weight)
* Evaluated weight 11 (attribute "sustainability")
17: 0: 026 (evaluate-weight)
availability: Prefer supplier06 > supplier02
* Evaluated weight 8 (attribute "availability")
18: 0: 07 (select-supplier)
** OUTPUT: Then try supplier06
19: 0: 03 (select-supplier)
** OUTPUT: Then try supplier02
20: 0: 08 (output-supplier-list)
*** DONE ***
```

S3

S4

The Power of Preferences

We coded this dynamic substate behavior with a single operator

- Using only 5 rules!

As you get more comfortable with Soar, you will often find that it can be much faster to code some logic routines with Soar rules than with traditional linear programming languages.

- Like all tools, each language has its own strengths

Congratulations! You have finished Lesson 07!

```
1: 0: 01 (init)
2: ==>S: S2 (operator tie)
3: 0: 015 (evaluate-weight)
total-cost: Prefer supplier03 > supplier01
total-cost: Prefer supplier04 > supplier01
total-cost: Prefer supplier05 > supplier01
total-cost: Prefer supplier03 > supplier02
total-cost: Prefer supplier04 > supplier02
total-cost: Prefer supplier05 > supplier02
total-cost: Prefer supplier03 > supplier06
total-cost: Prefer supplier04 > supplier06
total-cost: Prefer supplier05 > supplier06
* Evaluated weight 11.010000 (attribute "total-cost")
4: 0: 013 (evaluate-weight)
quality: Prefer supplier03 > supplier04
quality: Prefer supplier05 > supplier04
* Evaluated weight 11 (attribute "quality")
5: 0: 011 (evaluate-weight)
* Evaluated weight 8 (attribute "packaging")
6: 0: 010 (evaluate-weight)
speed: Prefer supplier05 > supplier03
* Evaluated weight 7 (attribute "speed")
7: 0: 06 (select-supplier)
** OUTPUT: First try supplier05
8: 0: 04 (select-supplier)
** OUTPUT: Then try supplier03
9: 0: 05 (select-supplier)
** OUTPUT: Then try supplier04
10: ==>S: S3 (operator tie)
11: 0: 022 (evaluate-weight)
* Evaluated weight 11.010000 (attribute "total-cost")
12: 0: 021 (evaluate-weight)
sustainability: Prefer supplier01 > supplier02
sustainability: Prefer supplier01 > supplier06
* Evaluated weight 11 (attribute "sustainability")
13: 0: 02 (select-supplier)
** OUTPUT: Then try supplier01
14: ==>S: S4 (operator tie)
15: 0: 029 (evaluate-weight)
* Evaluated weight 11.010000 (attribute "total-cost")
16: 0: 028 (evaluate-weight)
* Evaluated weight 11 (attribute "sustainability")
17: 0: 026 (evaluate-weight)
availability: Prefer supplier06 > supplier02
* Evaluated weight 8 (attribute "availability")
18: 0: 07 (select-supplier)
** OUTPUT: Then try supplier06
19: 0: 03 (select-supplier)
** OUTPUT: Then try supplier02
20: 0: 08 (output-supplier-list)
*** DONE ***
```