

Visualizing Soar

Mazin Assanie
mazina@umich.edu

Understanding This Takes Work

```
(S1 ^epmem E1 ^io I1 ^item2 not-subject ^item3 apple ^math 91 ^number 23
  ^object subject ^result1 subject ^result2 subject ^result3 imaginary
  ^result4 not-subject ^reward-link R1 ^smem S2 ^superstate nil ^svs S3
  ^type state)
(E1 ^command C1 ^present-id 1 ^result R2)
(I1 ^input-link I2 ^output-link I3)
(S2 ^command C2 ^result R3)
(S3 ^command C3 ^spatial-scene S4)
(S4 ^id world)

(S5 ^attribute state ^choices none ^epmem E2 ^impasse no-change
  ^intermediate apple ^intermediate-func 33 ^intermediate-na 33
  ^intermediate2 2 ^operator 01 ^operator 01 + ^quiescence t
  ^reward-link R4 ^smem S6 ^sub-number 90 ^sub-result1 imaginary
  ^sub-result2 imaginary ^sub-result3 imaginary ^superstate S1 ^svs S7
  ^thrown-out not-subject ^type state)
(E2 ^command C4 ^present-id 1 ^result R5)
(O1 ^item1 subject ^item2 subject)
(S6 ^command C5 ^result R6)
(S1 ^epmem E1 ^io I1 ^item2 not-subject ^item3 apple ^math 91 ^number 23
  ^object subject ^result1 subject ^result2 subject ^result3 imaginary
  ^result4 not-subject ^reward-link R1 ^smem S2 ^superstate nil ^svs S3
  ^type state)
(E1 ^command C1 ^present-id 1 ^result R2)
(I1 ^input-link I2 ^output-link I3)
(S2 ^command C2 ^result R3)
(S3 ^command C3 ^spatial-scene S4)
(S4 ^id world)
(S7 ^command C6 ^spatial-scene S8)
(S8 ^id world)
```

Understanding This Takes Work

```

- {
2: (<c1> ^retrieved-stack <r*4>) ([1188] ^retrieved-stack [1142]) i 177 -> i 180 -> i 182 -> i 193
3: (<r*4> ^item <i*1>) ([1142] ^item [1143]) i 177 -> i 180 -> i 182 -> i 193
4: (<i*1> ^lt <l*1>) ([1143] ^lt [1144]) i 177 -> i 180 -> i 182 -> i 193
5: (<l*1> ^spelling *) ([1144] ^spelling *) i 177 -> i 180 -> i 182 -> i 193
}
6: (<c1> ^prior-word <f1>) ([1188] ^prior-word [1140]) i 177 -> i 180 -> i 182 -> i 193
7: (<f1> ^spelling *) ([1140] ^spelling *) i 177 -> i 180 -> i 182 -> i 193
8: (<s1> ^operator <o1>) ([1118] ^operator [1119]) i 180 -> i 182 -> i 193
9: (<o1> ^name comprehend-word) ([1119] ^name comprehend-word) i 175 -> i 202
10: (<o1> ^current-word <w1>) ([1119] ^current-word [1268]) i 175 -> i 202
- {
11: (<c1> ^prior-word <p*1>) ([1188] ^prior-word [1189]) i 182 -> i 193
12: (<p*1> ^spelling *) ([1189] ^spelling *) i 182 -> i 193
13: (<w1> ^next <r*2>) ([1268] ^next [1190]) i 182 -> i 193
14: (<r*2> ^spelling .) ([1190] ^spelling .) i 182 -> i 193
}
15: (<w1> ^first-word true) ([1268] ^first-word true) i 202
16: (<w1> ^spelling all) ([1268] ^spelling all) i 189 -> i 193
->
1: (<c1> ^comprehension-structure <c2> +) ([1327] ^comprehension-structure <c2> +)
2: (<c2> ^processed true +) ([1122] ^processed true +)
3: (<c2> ^type word +) ([1233] ^type [1236] +)
4: (<c2> ^retrieved-lexical-item @A10085 +) ([1286] ^retrieved-lexical-item [ (1271) ] +)
5: (@A10085 ^super-converts-to CP +) ([1281] ^super-converts-to [1282] +)
6: (@A10085 ^specifier universal +) (@A10085 ^specifier universal +)
7: (@A10085 ^referent @R10076 +) (@A10085 ^referent @R10076 +)
8: (@R10076 ^handle all +) (@R10076 ^handle all +)
9: (@A10085 ^structure-type QUANT +) (@A10085 ^structure-type QUANT +)
10: (@A10085 ^spelling all +) (@A10085 ^spelling all +)
11: (@A10085 ^number plural +) (@A10085 ^number plural +)
12: (<c2> ^not-merged-receiver <i1> +) (<c2> ^not-merged-receiver <i1> +)
13: (<i1> ^first-word true +) ([1281] ^first-word [1276] +)
14: (<i1> ^semantics <s2> +) ([1281] ^semantics [1279] +)
15: (<i1> ^lt @A10085 +) ([1294] ^lt [ (1271) ] +)
16: (<i1> ^current-word <w1> +) ([1319] ^[1322] [1323] +)
17: (<i1> ^structure-type QUANT +) ([1306] ^structure-type [ (1296) ] +)
18: (<i1> ^lt-referent <n1> +) ([1280] ^lt-referent [1281] +)
19: (<n1> ^handle all +) ([1328] ^[ (1322) ] [ (1323) ] +)

```

Motivation

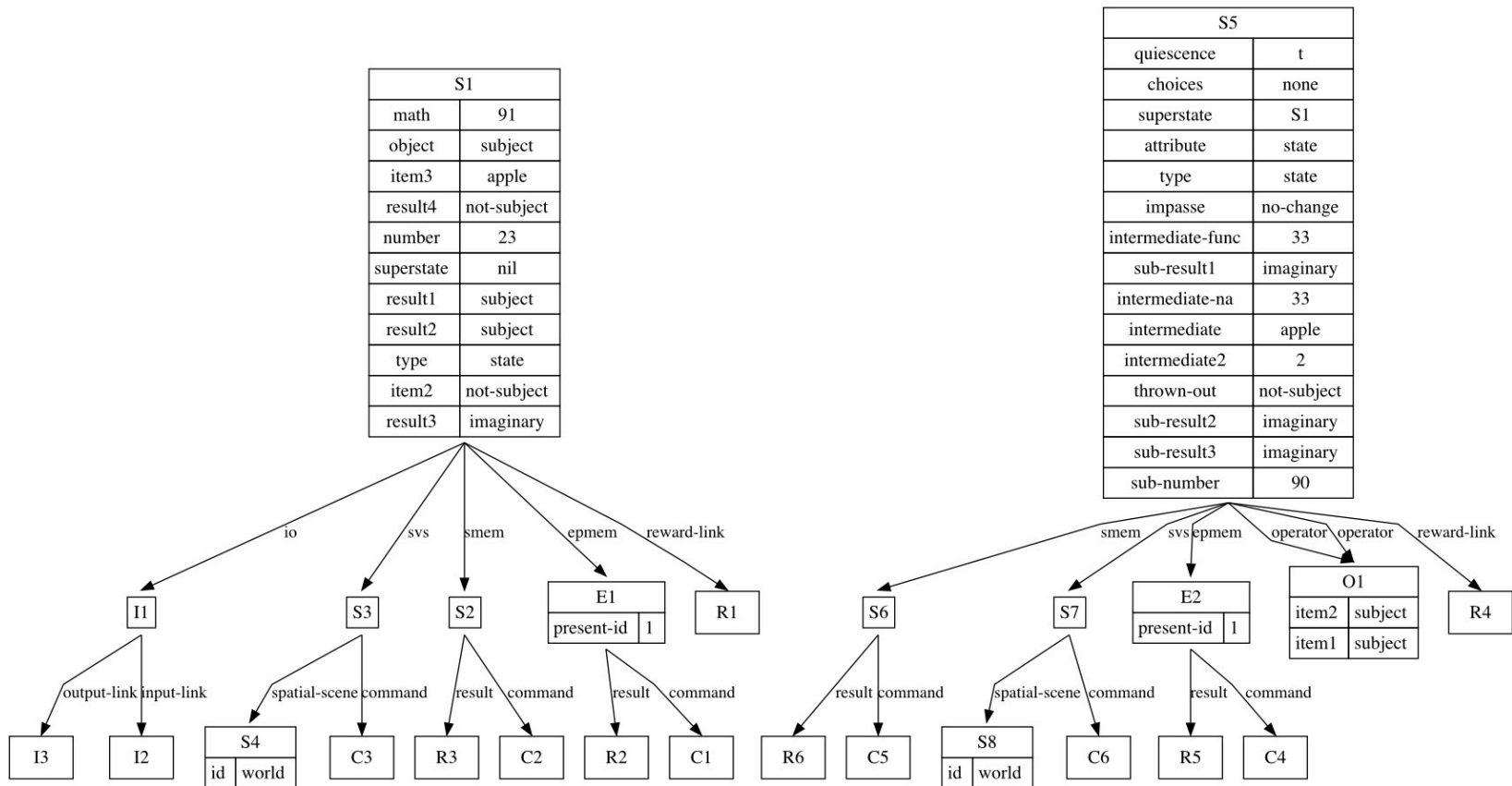
- Visually parsing Soar's output is often very cumbersome, especially with complex agents.
- A visual representation can:
 - Present more information in a compact, intuitive way
 - Make it easier to examine memory and/or find something you're looking for
 - May allow you to find patterns you may not have otherwise noticed

Example (Working Memory)

```
(S1 ^epmem E1 ^io I1 ^item2 not-subject ^item3 apple ^math 91 ^number 23
  ^object subject ^result1 subject ^result2 subject ^result3 imaginary
  ^result4 not-subject ^reward-link R1 ^smem S2 ^superstate nil ^svs S3
  ^type state)
(E1 ^command C1 ^present-id 1 ^result R2)
(I1 ^input-link I2 ^output-link I3)
(S2 ^command C2 ^result R3)
(S3 ^command C3 ^spatial-scene S4)
(S4 ^id world)

(S5 ^attribute state ^choices none ^epmem E2 ^impasse no-change
  ^intermediate apple ^intermediate-func 33 ^intermediate-na 33
  ^intermediate2 2 ^operator 01 ^operator 01 + ^quiescence t
  ^reward-link R4 ^smem S6 ^sub-number 90 ^sub-result1 imaginary
  ^sub-result2 imaginary ^sub-result3 imaginary ^superstate S1 ^svs S7
  ^thrown-out not-subject ^type state)
(E2 ^command C4 ^present-id 1 ^result R5)
(O1 ^item1 subject ^item2 subject)
(S6 ^command C5 ^result R6)
(S1 ^epmem E1 ^io I1 ^item2 not-subject ^item3 apple ^math 91 ^number 23
  ^object subject ^result1 subject ^result2 subject ^result3 imaginary
  ^result4 not-subject ^reward-link R1 ^smem S2 ^superstate nil ^svs S3
  ^type state)
(E1 ^command C1 ^present-id 1 ^result R2)
(I1 ^input-link I2 ^output-link I3)
(S2 ^command C2 ^result R3)
(S3 ^command C3 ^spatial-scene S4)
(S4 ^id world)
(S7 ^command C6 ^spatial-scene S8)
(S8 ^id world)
```

Example (Working Memory)



What can be visualized

- What can be visualized?
 - Three Soar memory systems:
 - Working memory
 - Semantic memory
 - Episodic memory
 - Chunks and instantiations recorded by the explainer
 - Two complex explainer graphs
 - Each compress a massive amount of explainer data into a single image

Usage

- To visualize memory:

```
visualize [wm | smem | epmem] [id]
```

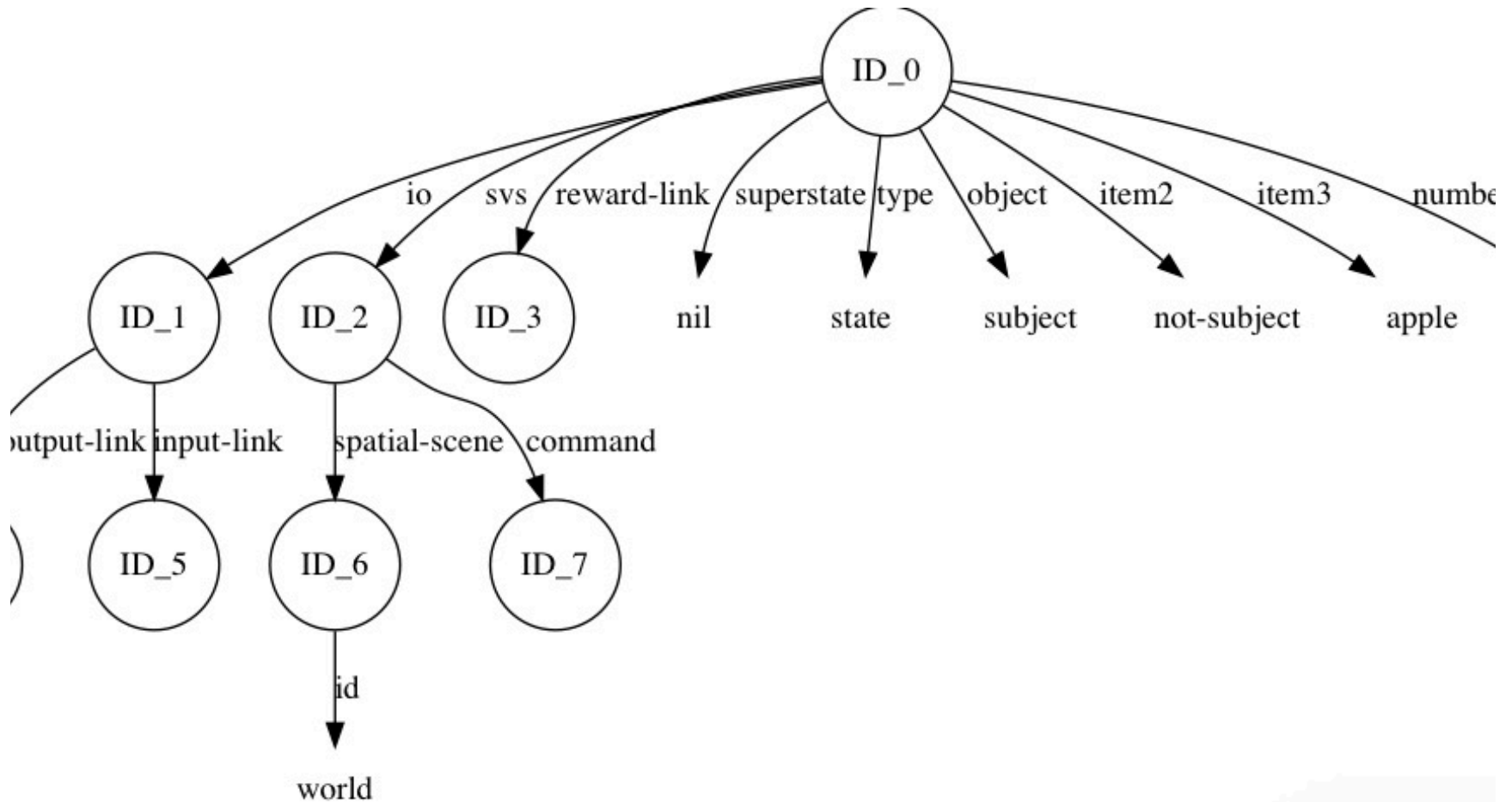
- To visualize explainer output:

```
visualize [explainer| instantiations | contributors]
```

Notes:

- The command saves an image to disk and immediately shows image in viewer/editor
- Command provides shared options

Example (Episodic/Semantic Memory)



3 Explainer Visualizations

1. Instantiation Graph:

- All instantiations involved in formation of discussed chunk
- Conditions in one rule are connected to the actions that created the WME they tested

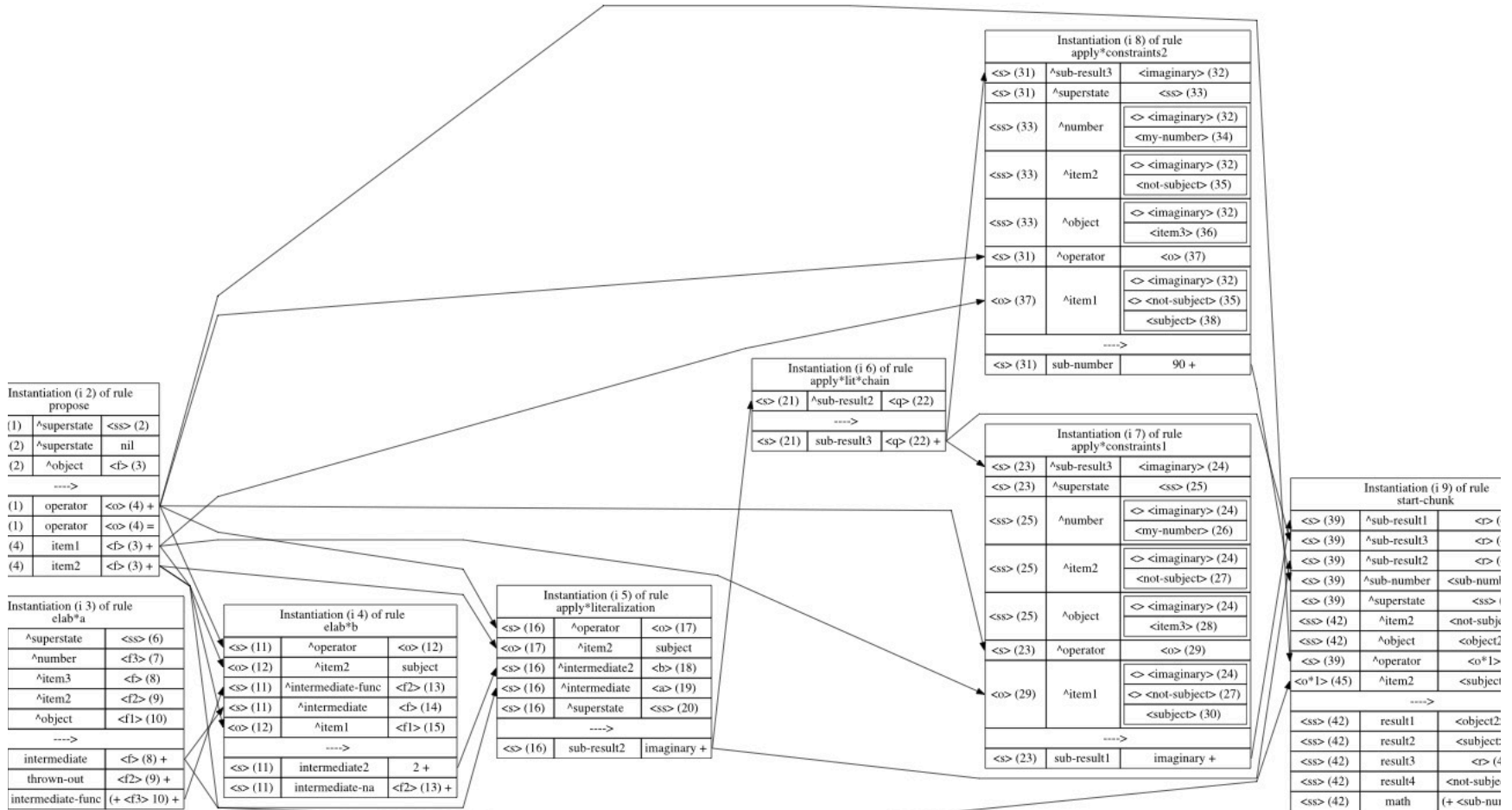
2. Chunk contribution graph:

- Discussed chunk + instantiation graph
- Conditions in chunk are connected to the conditions in the instantiation graph upon which they were based.

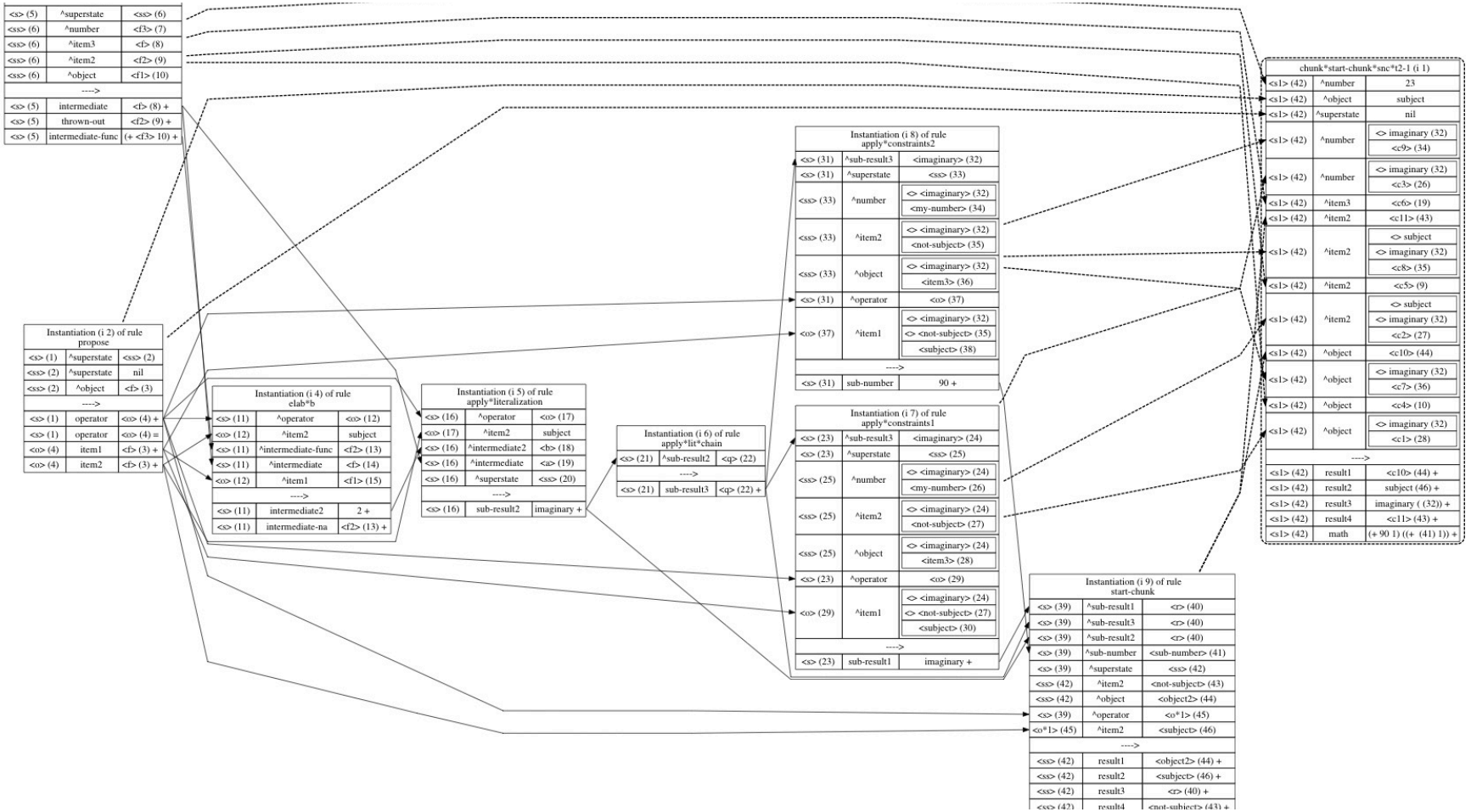
3. Single Rule: Last chunk or instantiation explained

Note: Visualizer will use whichever trace the explainer is using.

Example (Instantiation Graph)



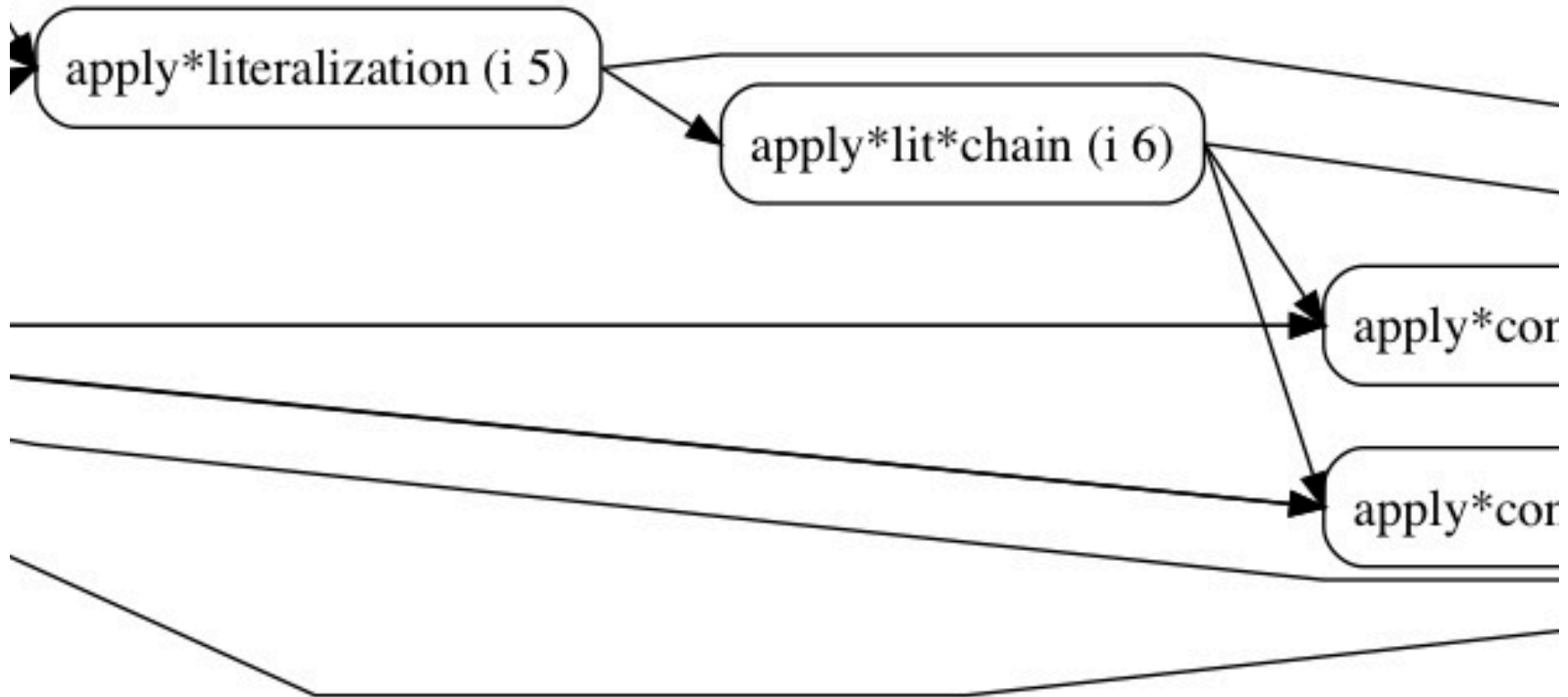
Example (Chunk Contribution Graph)



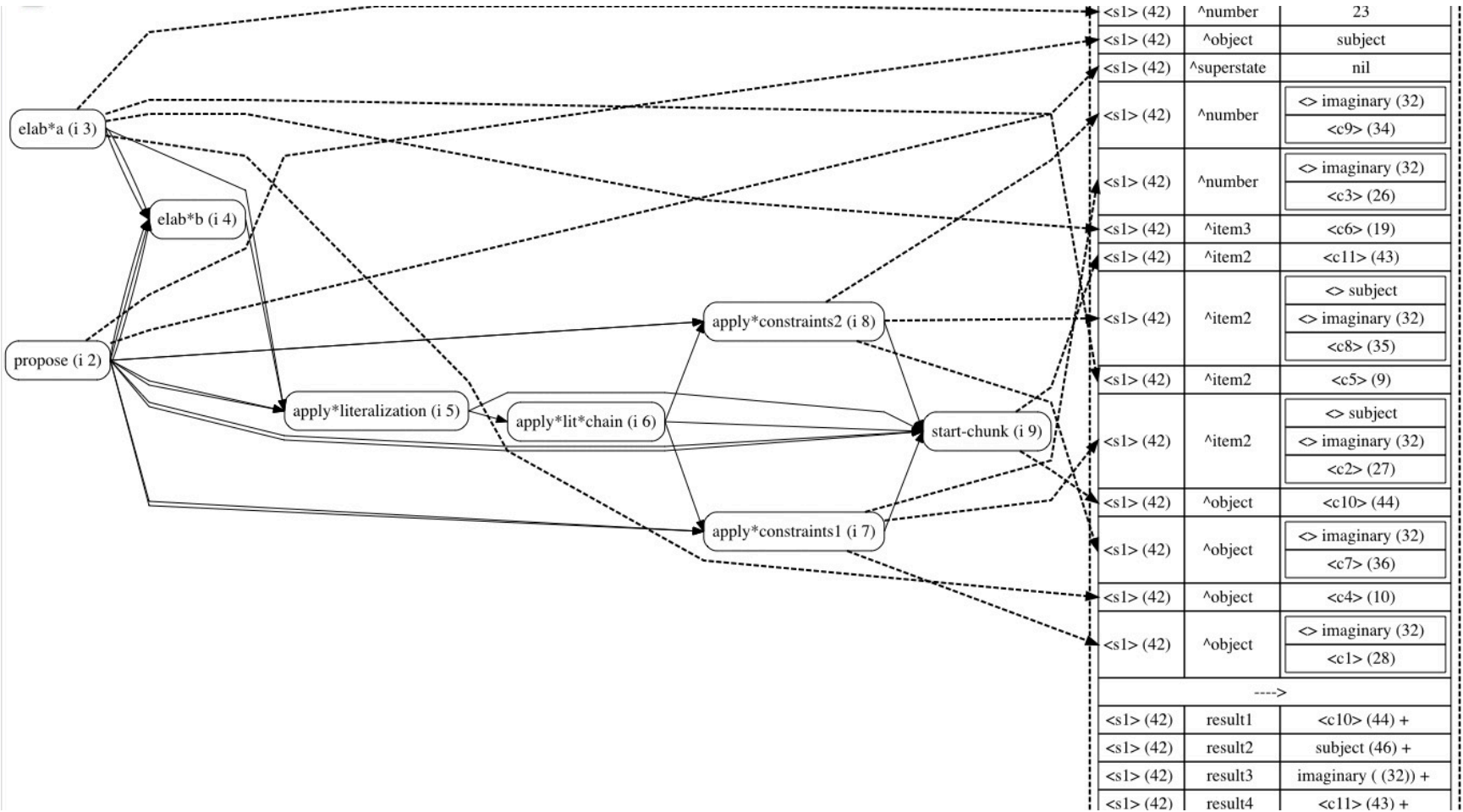
One Important Option

- **--simple-style** [yes | no]
 - For memory systems, this controls whether to print memory as “records” or a standard graph
 - For explainer graphs, this controls whether to print just the rule name or the full conditions and actions

Example (Instantiation Graph)



Example (Chunk Contributors)



Nuggets

- Chunk explanation are way easier to understand visually
- “Record” view of WM and rules were surprisingly clear
- Ability to import and edit visualization very useful for making figures for papers and presentation slides
 - Omnigraffle plug
- Memory visualization pane in graphical debugger?

Coals

- Automatic layout of complex graphs doesn't always work well.
 - Will experiment with more settings
 - You may need to move things around in editor.
- Automatically launching a viewer or the editor uses a system call that may not be available on all systems.