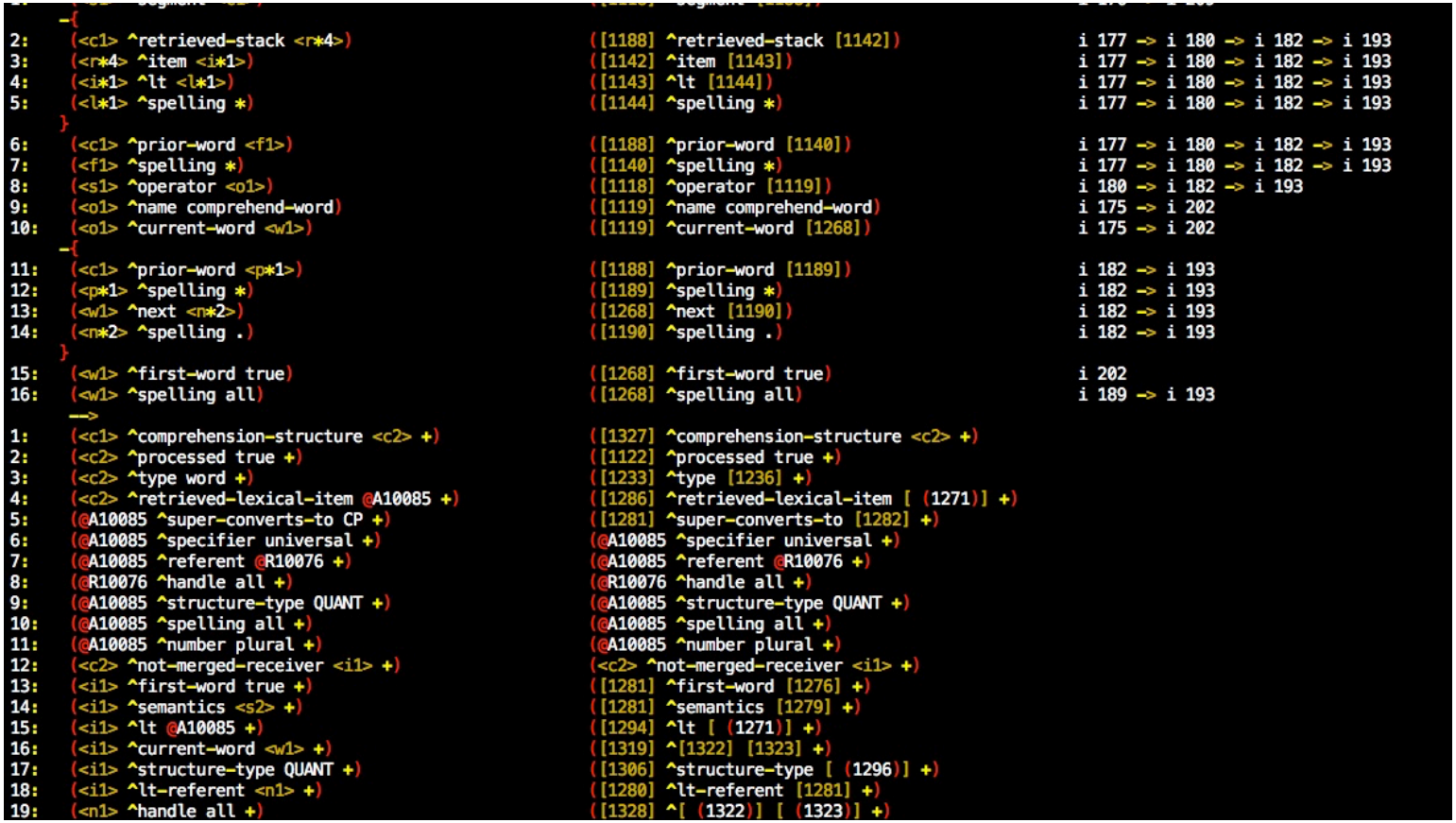
Visualizing Soar

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 O1 ^operator O1 + ^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



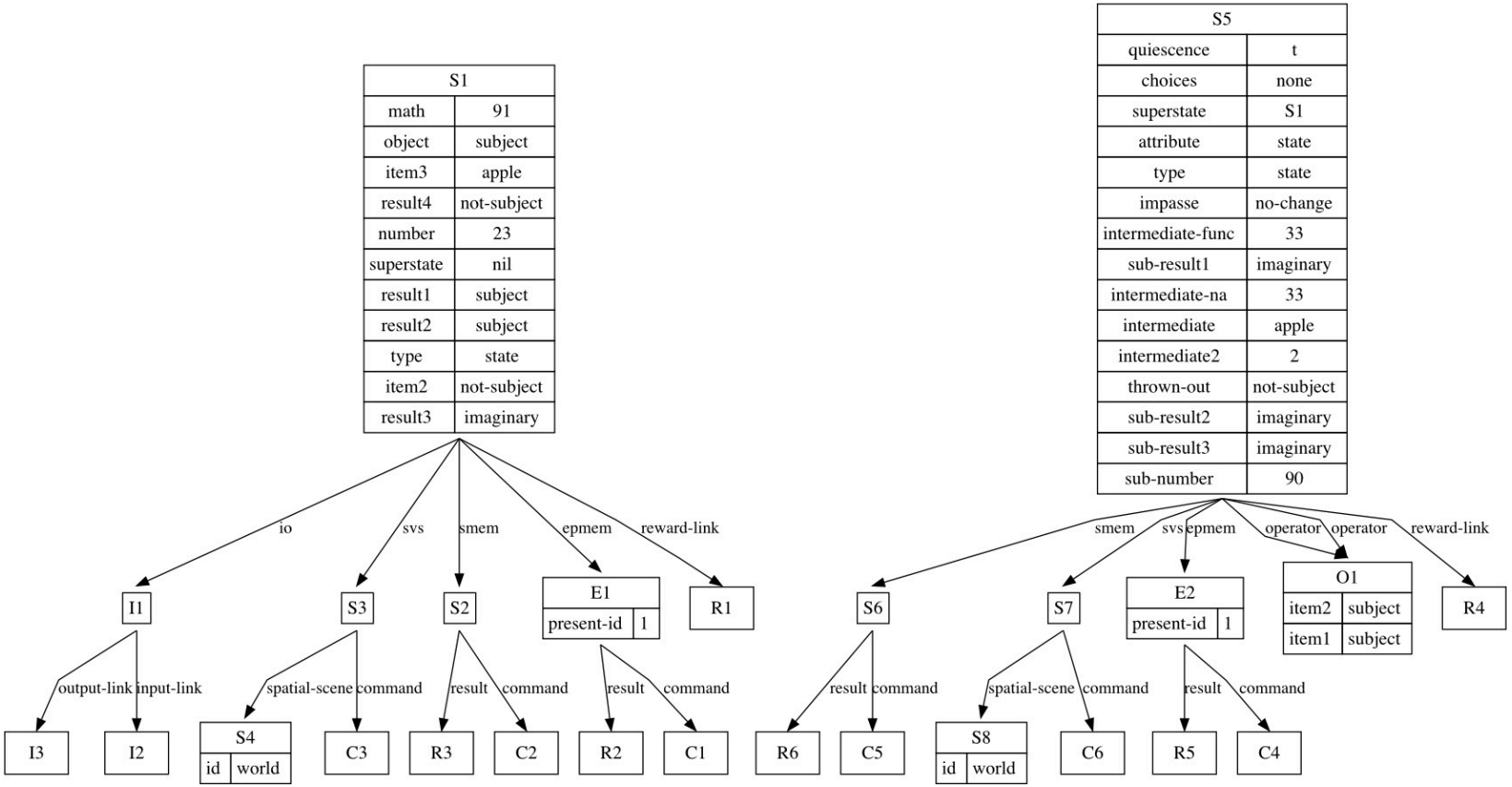
Motivation

- 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 O1 ^operator O1 + ^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]
```

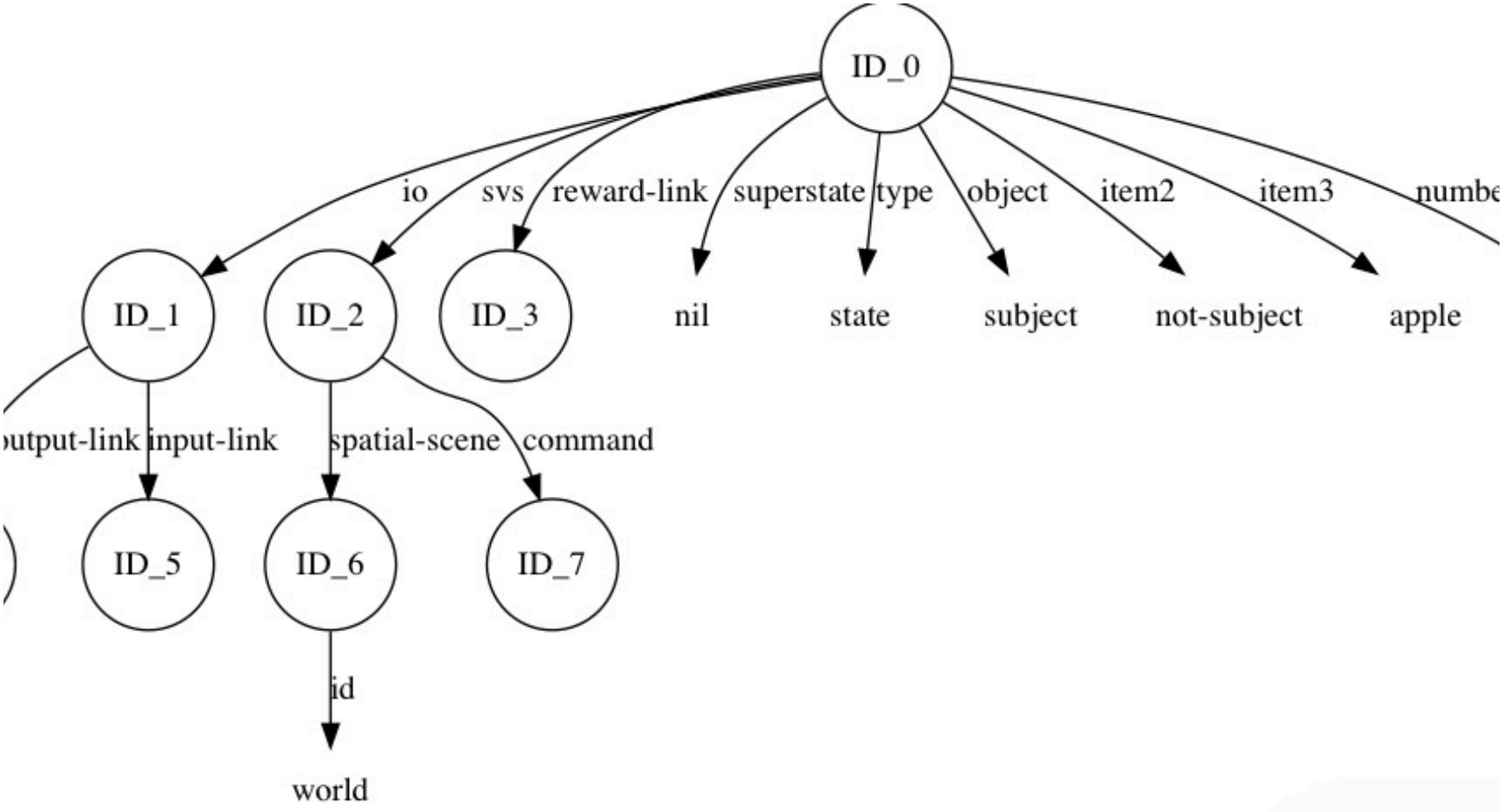
- To visualize explainer output:

```
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)

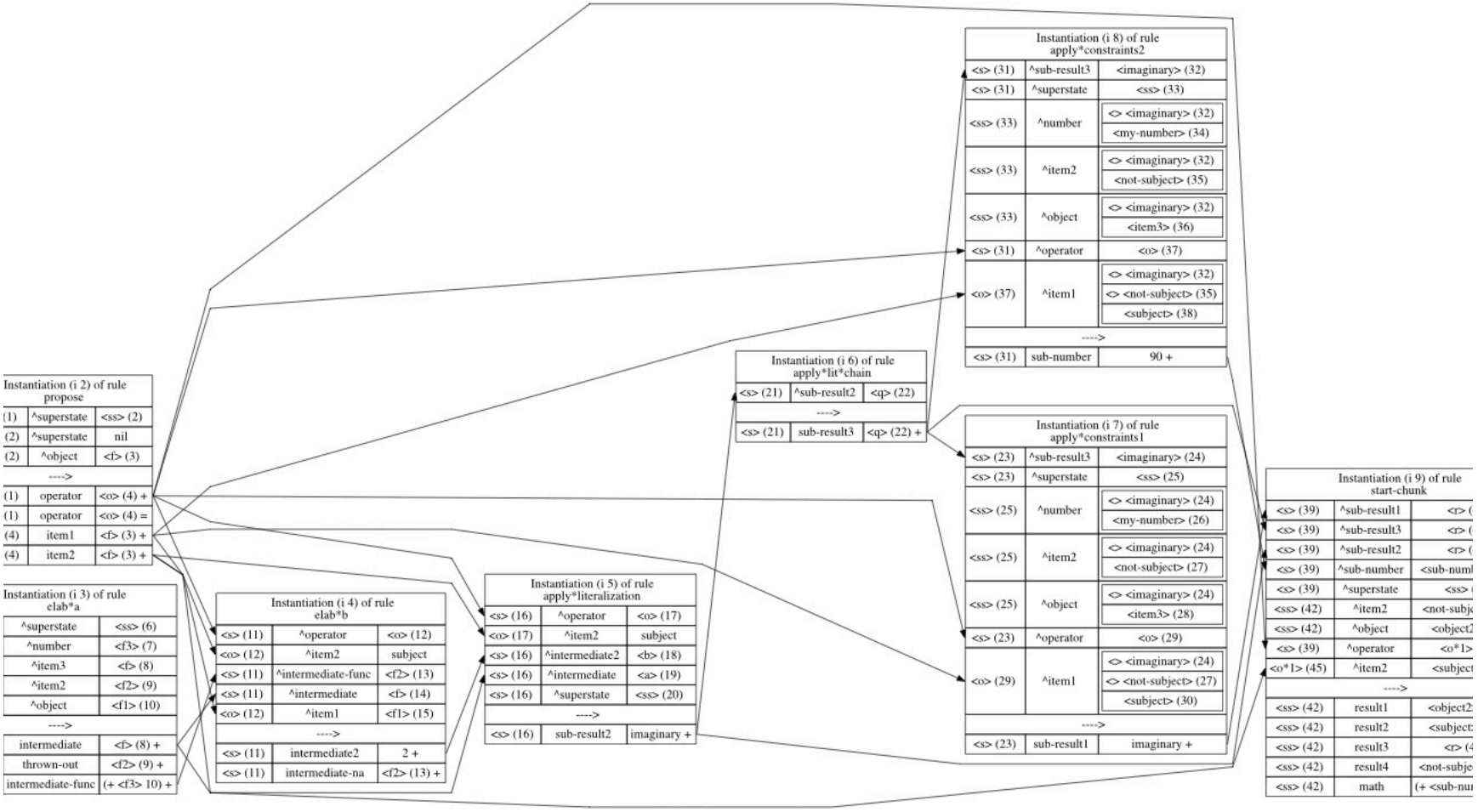


3 Explainer Visualizations

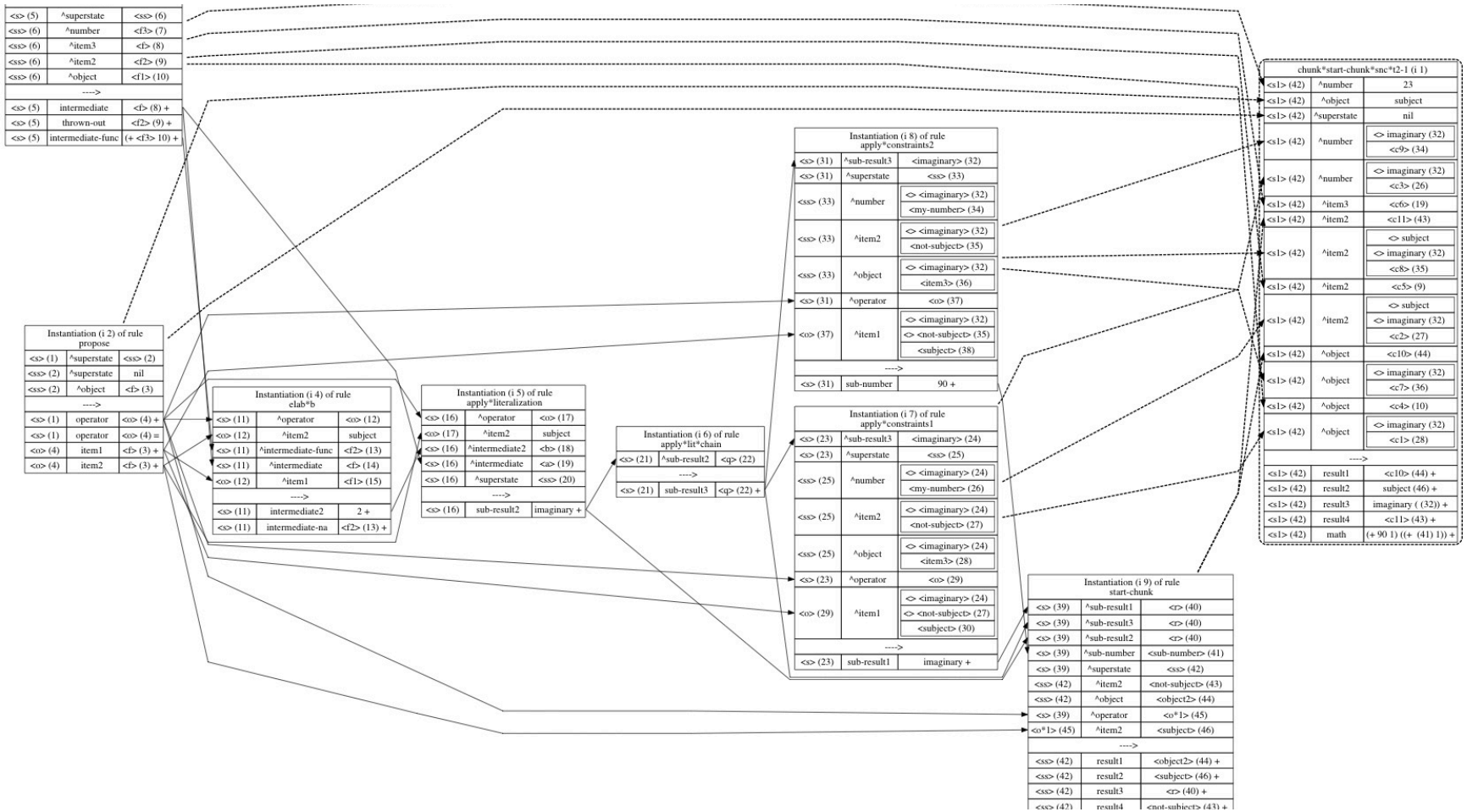
1. Instantiation Graph:
   - All instantiations involved in formation of discussed chunk
   - Conditions in one rule are connected to the actions that created the WME they tested
2. Chunk contribution graph:
   - Discussed chunk + instantiation graph
   - Conditions in chunk are connected to the conditions 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)



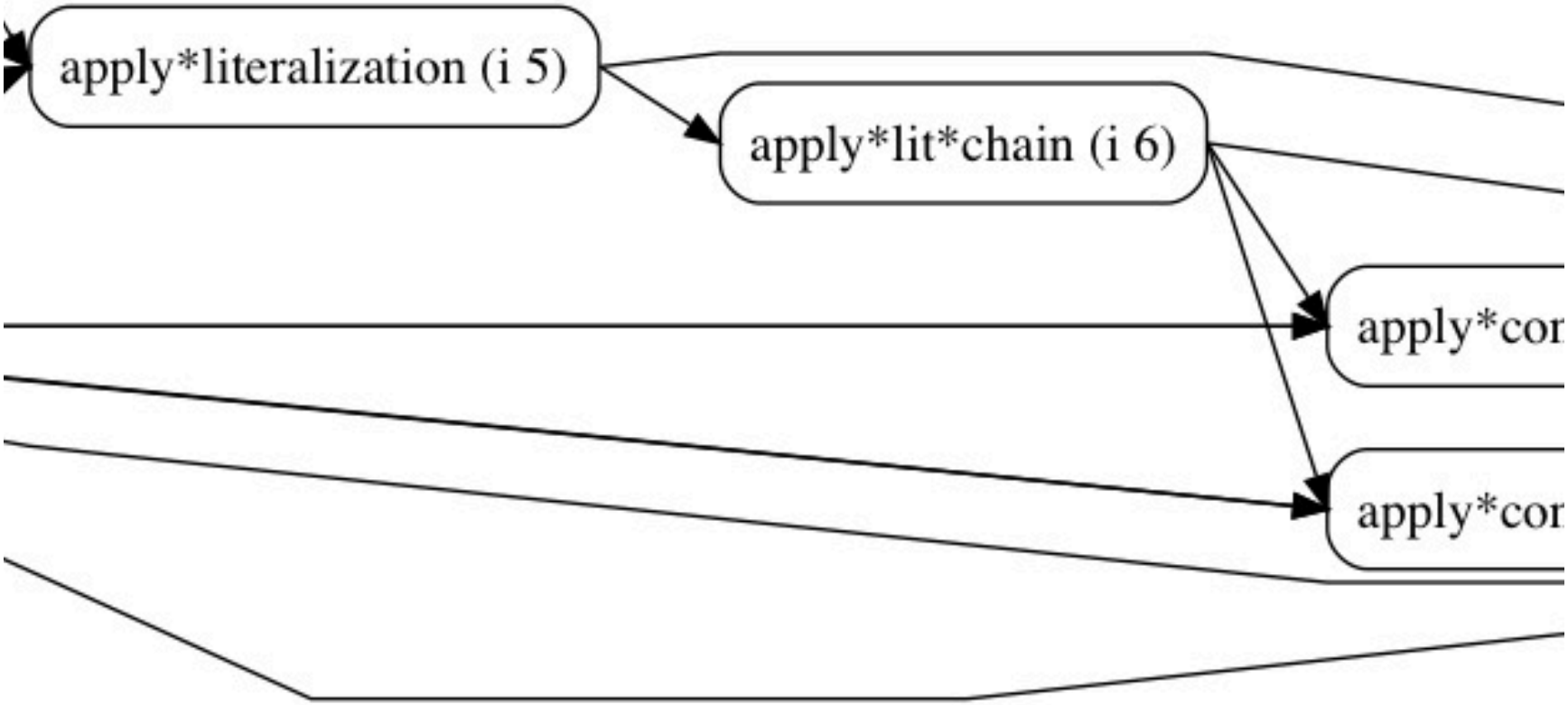
Example (Chunk Contribution Graph)



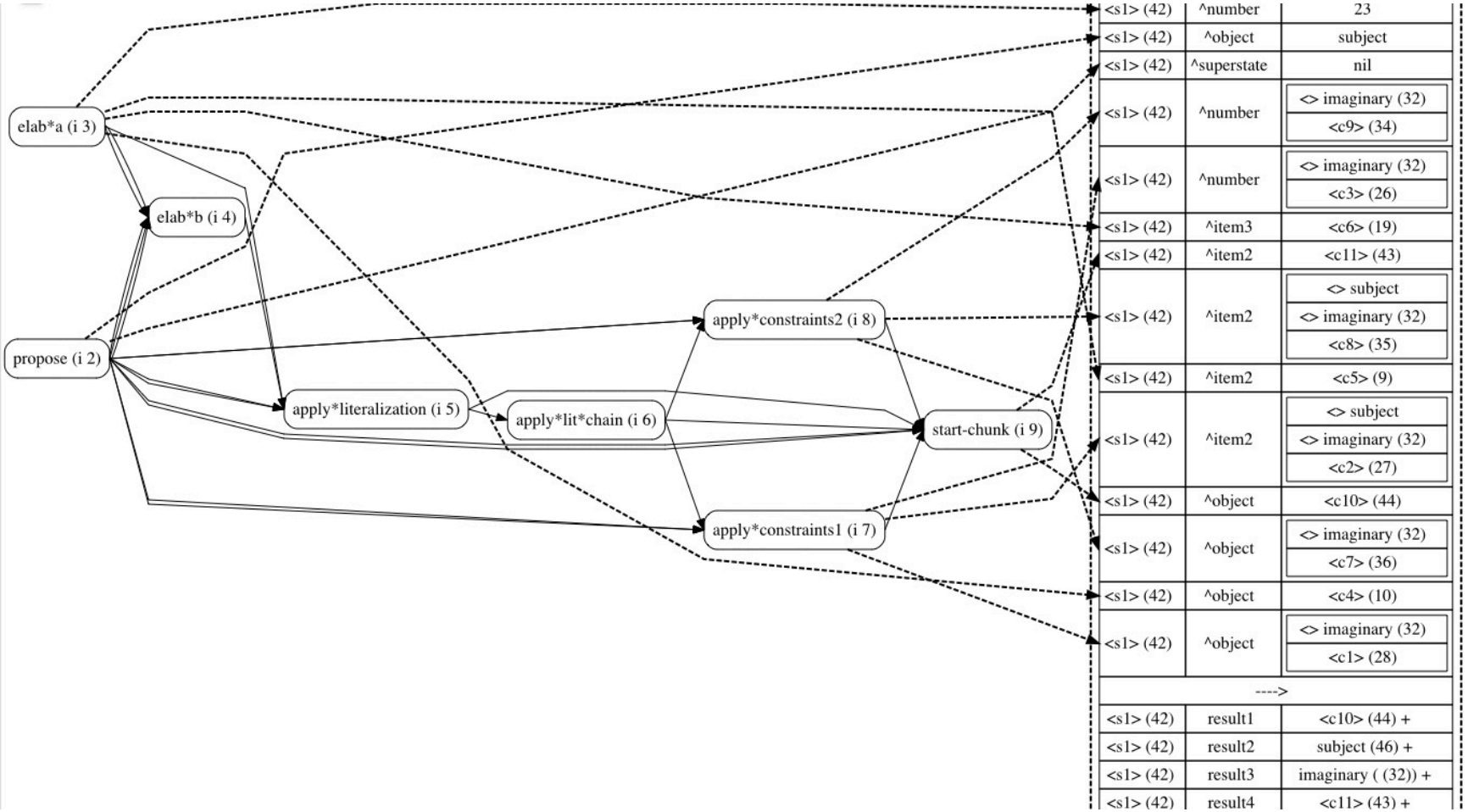
One Important Option

- **--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.