Soar Basics

Soar Tutorial
May 6, 2019

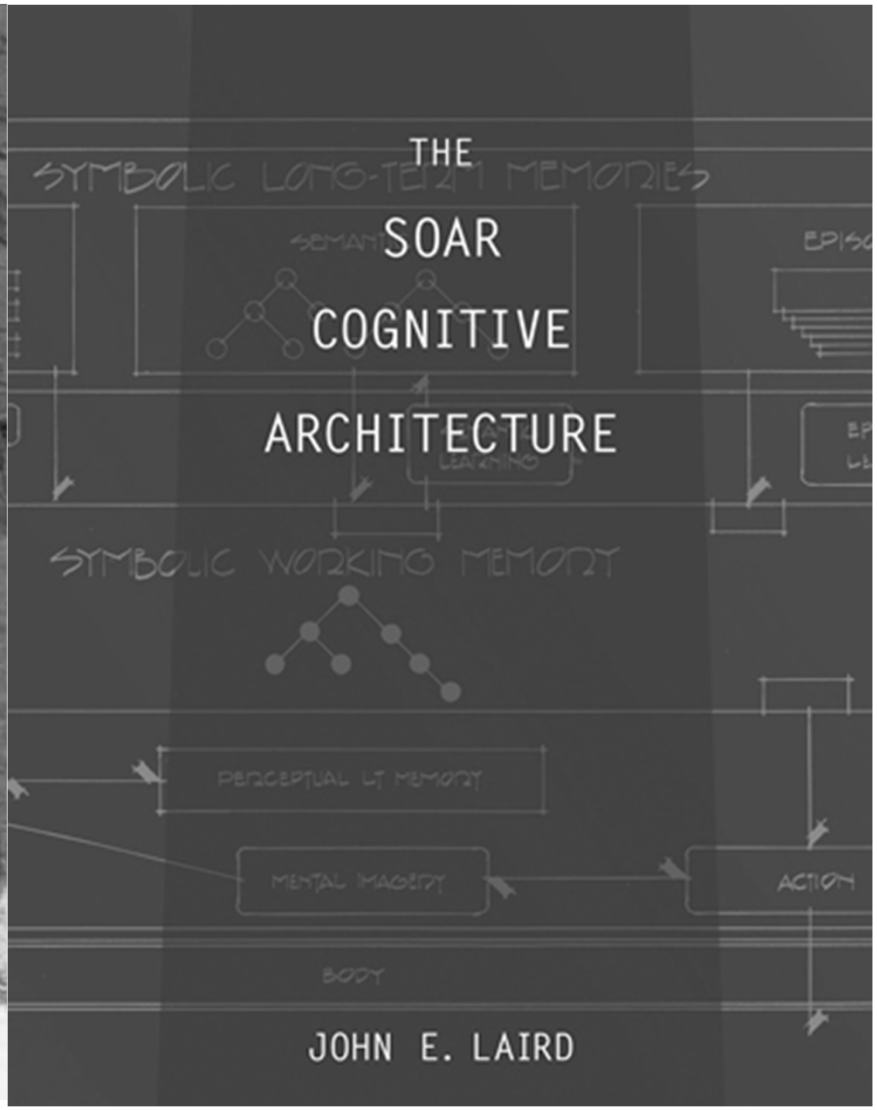
READ THE BOOK!

READ THE MANUAL!

The Soar Cognitive Architecture (Laird, Newell, Rosenbloom, et al.; 1981-)

- Goal: knowledge-rich, long-living autonomous agents that interact with humans and the world in real time
  - Over 35 years of development
- Inspired by psychology and biology
  - Look to psychology for useful cognitive mechanisms and capabilities
  - Look to computer science and AI for efficient and robust implementations
- More complex behavior and tasks, longer time scales.
  - Integrated hierarchical planning and execution
  - Episodic memory and mental imagery
  - Large bodies of knowledge, multiple types of learning
  - Faster than real-time execution over hours of execution
    - Procedural cycle time < .3 ms.
- Available on all major platforms: Windows, Linux
  - Open source (BSD) including tools and agents
  - More than 100 systems implemented in Soar

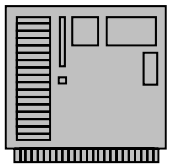
The Soar Cognitive Architecture
(Laird, Newell, Rosenbloom, et al.; 1981-)



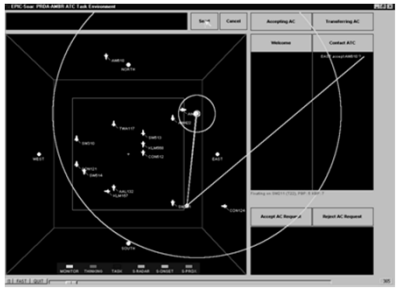
Research Methodology

- Maintain a single integrated architectural implementation
  - Force everything (and everyone) to work together
- Explore new architectural mechanisms or extensions to existing mechanisms
  - Episodic memory, semantic memory, reinforcement learning, mental imagery, emotion-influenced processing
  - Develop implementations that scale to large knowledge bases and with minimal computational overhead.
- New cognitive capabilities that exploit architectural mechanisms
- Choose tasks that require integration of many architectural and cognitive capabilities, and involve large bodies of knowledge

Example Virtual Environments

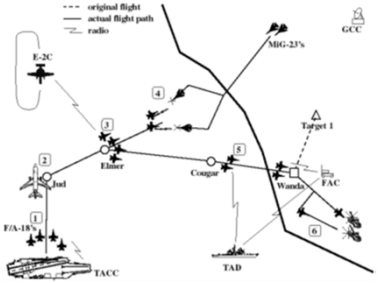


R1-Soar
Computer Configuration



Amber EPIC-Soar
Modeling Human-Computer Interaction

ICT Virtual Human
Natural Interaction, Emotion

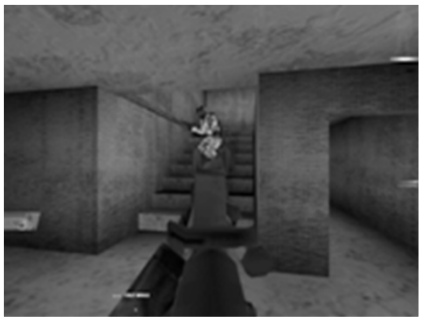


TacAir/RWA-Soar
Complex Doctrine & Tactics

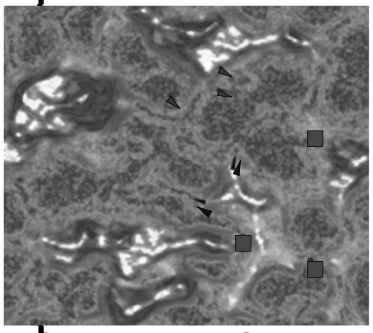
Soar Quakebot
Anticipation

StarCraft
Spatial Reasoning & Real-time Strategy

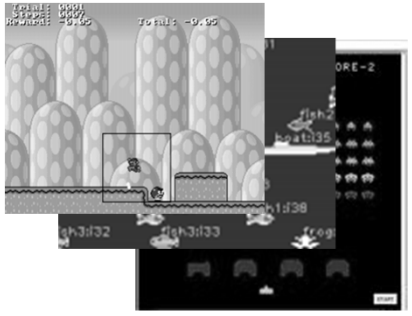
Haunt
AI Actors and Director



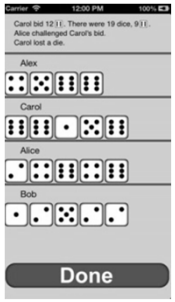
MOUTbot
Team Tactics



Simulated Scout
Spatial Reasoning & Mental Imagery



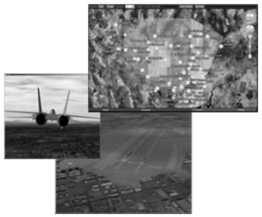
Action Games
Spatial Reasoning & Reinforcement Learning



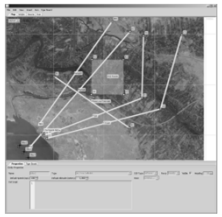
Liar's Dice
Probabilistic reasoning and reinforcement learning

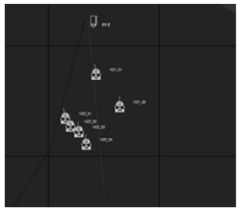
Viewpoints
Creative Human Interaction

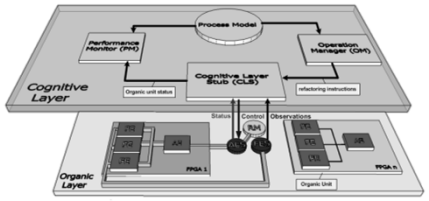
Example Environments from Soar Technology


RedRef

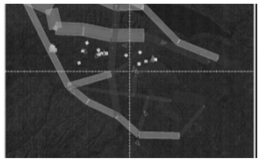

EDGE

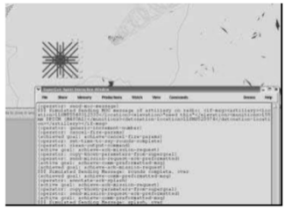

TigerBoard

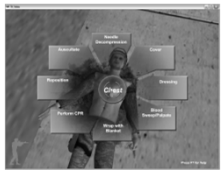

AutoWingman

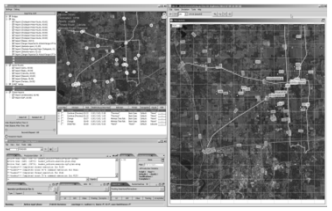


Soar Longevity

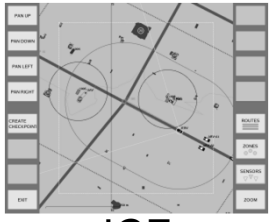

AutoATC


SAGIS

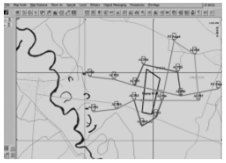

ISAT

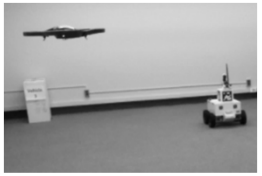

SUMET

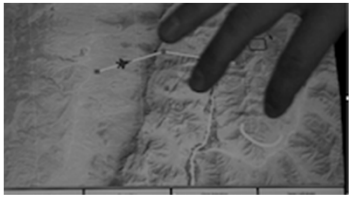

CCA


ICF


CERES


ECGF


SID Cargo UAV


SID Mav

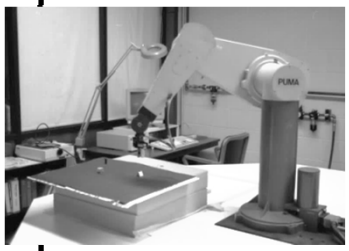

SID UGV


SID MAGIC

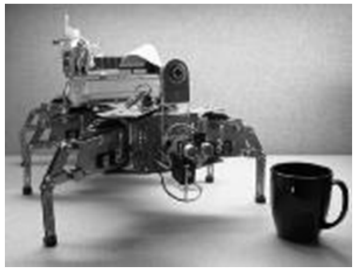

AGILE


JFETS

Soar Robotic Platforms



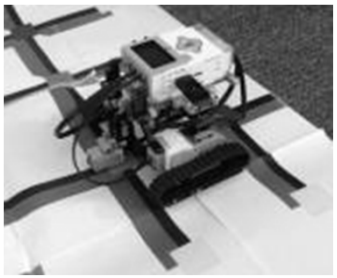
1988: Robo-Soar, UM



2009: Penn State

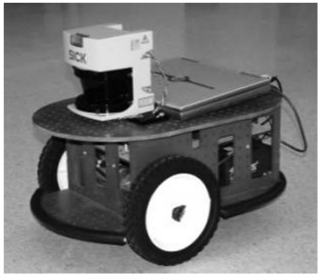
2011: Superdroid, PSU

2012: BOLT, UM/ST



2014: Mindstorms, UM

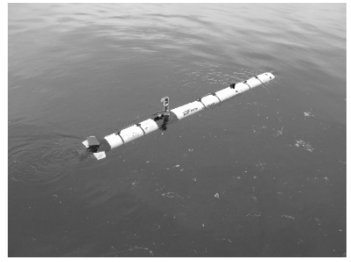
1990: Hero-Soar, UM



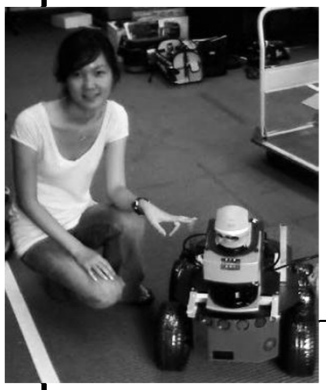
2009: Splinter, UM

2011: Magic, ST

2013: REEM-C
Pal Robotics



2015: Penn State



2004: Adapt, Pace U

2010: Soar Tech

2012: rGator, ST

2013: Summit, ST

2015: Magic 2, UM

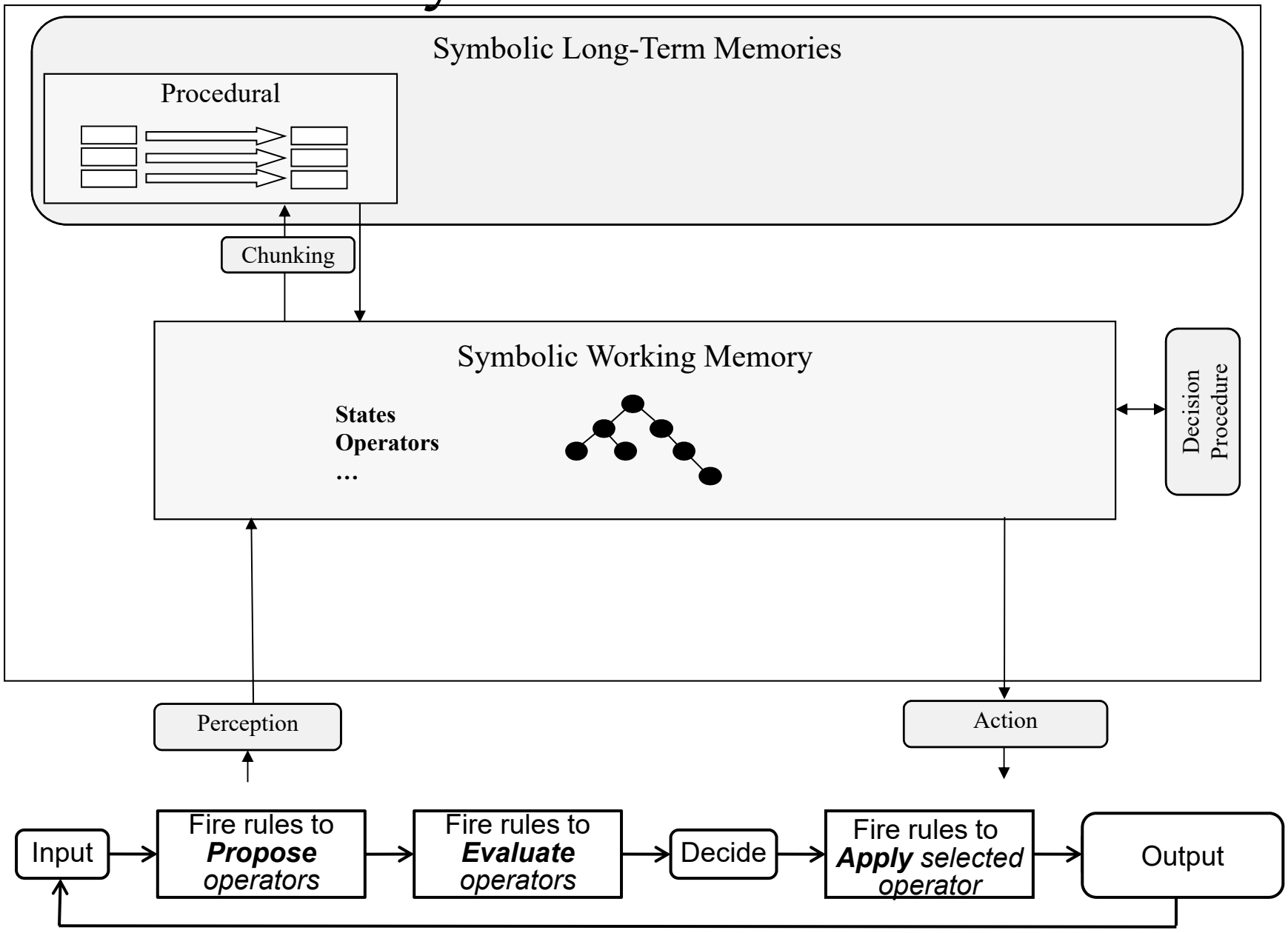
Interactive Task Learning

- Learn new tasks through natural interactions with humans
  - Concept definitions, hierarchical goal descriptions, failure states, task constraints, task actions, heuristics, procedures, …
- Rosie (Soar)
  - Pre-encoded procedural and semantic knowledge implements task learning strategy
  - *No new learning mechanisms*
  - Learns >50 puzzles, games, and mobile robot tasks

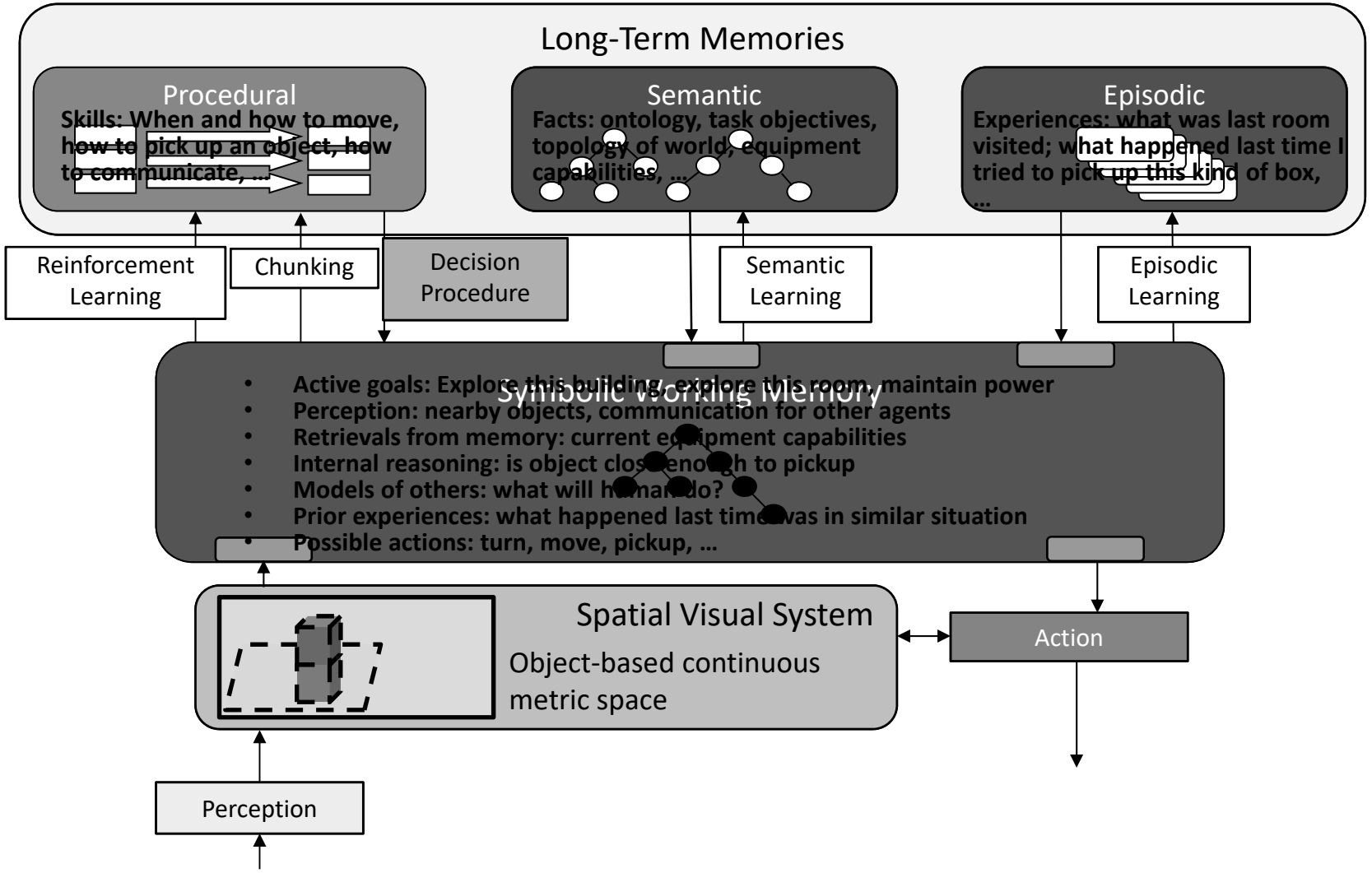
VIDEO

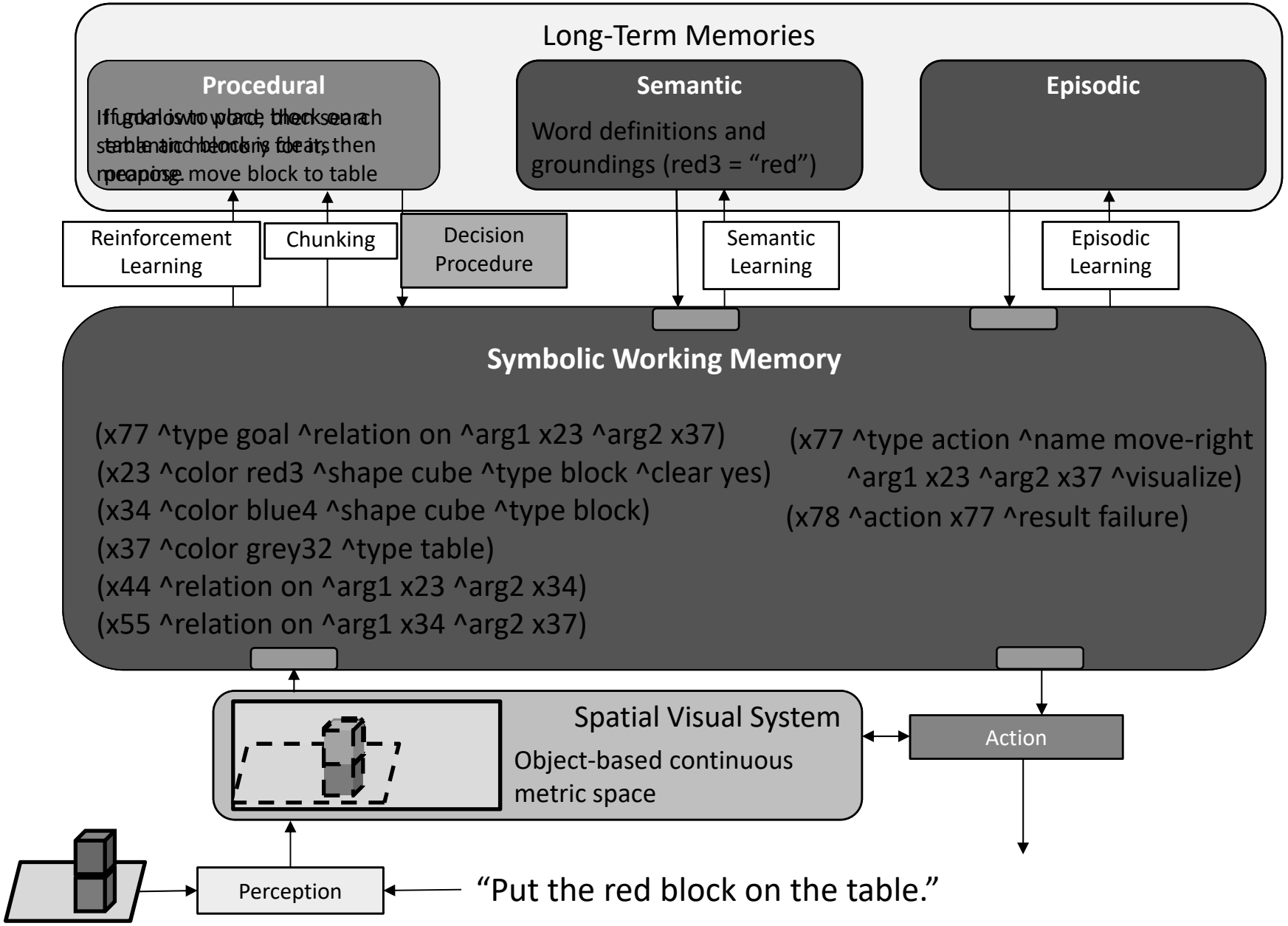
James Kirk

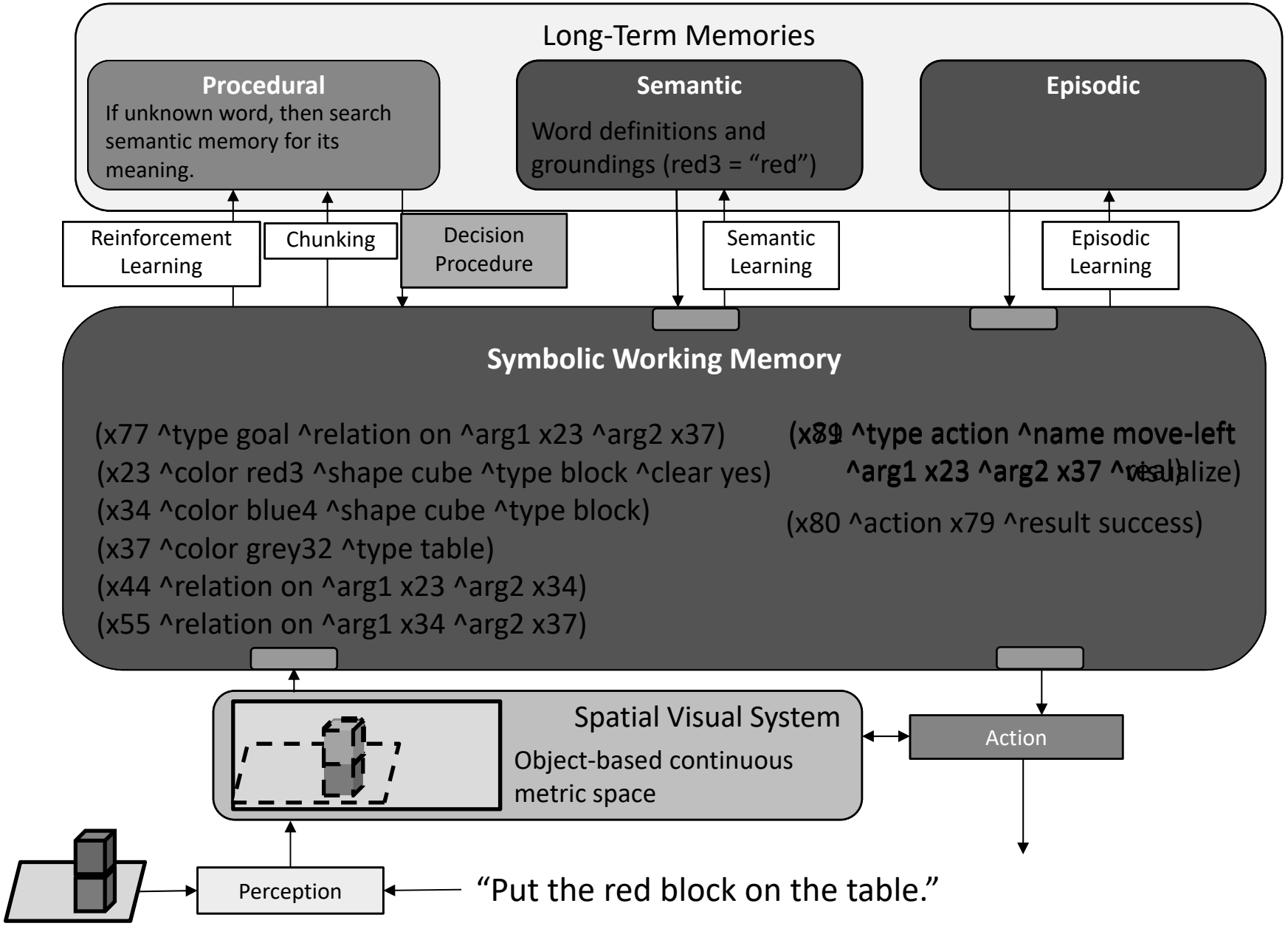
Early Soar Structure



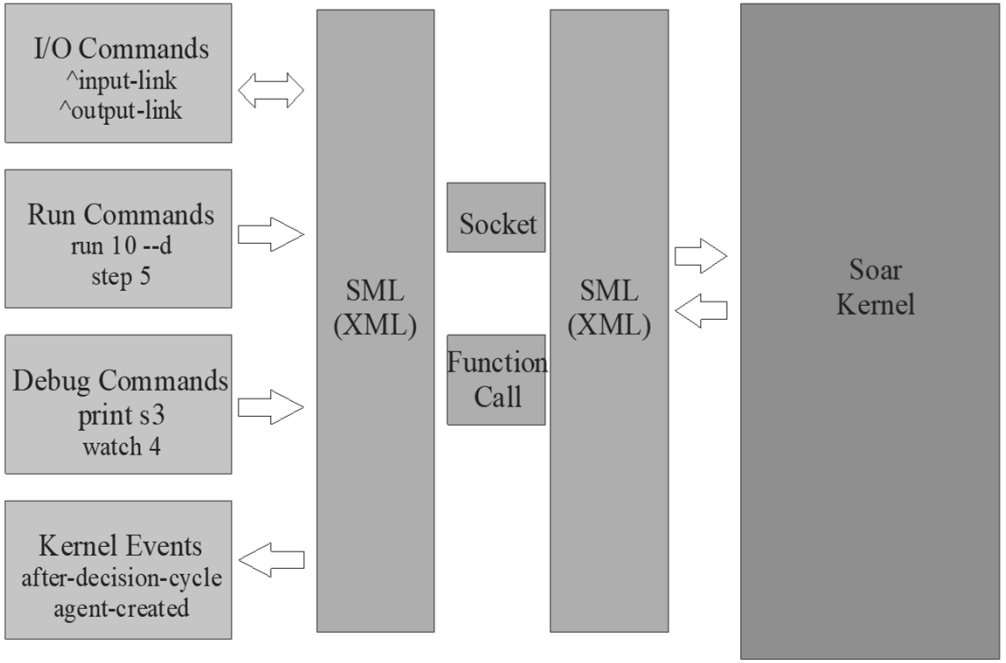
Soar 9 Structure




Long-Term Memories
Procedural
If goal is to place, then search
semantic memory for it,
then propose move block to table
Semantic
Word definitions and
groundings (red3 = "red")
Episodic
Reinforcement Learning
Chunking
Decision Procedure
Semantic Learning
Episodic Learning
Symbolic Working Memory
(x77 ^type goal ^relation on ^arg1 x23 ^arg2 x37)
(x23 ^color red3 ^shape cube ^type block ^clear yes)
(x34 ^color blue4 ^shape cube ^type block)
(x37 ^color grey32 ^type table)
(x44 ^relation on ^arg1 x23 ^arg2 x34)
(x55 ^relation on ^arg1 x34 ^arg2 x37)
(x77 ^type action ^name move-right
^arg1 x23 ^arg2 x37 ^visualize)
(x78 ^action x77 ^result failure)
Spatial Visual System
Object-based continuous
metric space
Action
Perception
"Put the red block on the table."


Long-Term Memories
Procedural
If unknown word, then search semantic memory for its meaning.
Semantic
Word definitions and groundings (red3 = "red")
Episodic
Reinforcement Learning
Chunking
Decision Procedure
Semantic Learning
Episodic Learning
Symbolic Working Memory
(x77 ^type goal ^relation on ^arg1 x23 ^arg2 x37)
(x23 ^color red3 ^shape cube ^type block ^clear yes)
(x34 ^color blue4 ^shape cube ^type block)
(x37 ^color grey32 ^type table)
(x44 ^relation on ^arg1 x23 ^arg2 x34)
(x55 ^relation on ^arg1 x34 ^arg2 x37)
(x89 ^type action ^name move-left ^arg1 x23 ^arg2 x37 ^result)
(x80 ^action x79 ^result success)
Spatial Visual System
Object-based continuous metric space
Action
Perception
"Put the red block on the table."

Integration with an External Environment



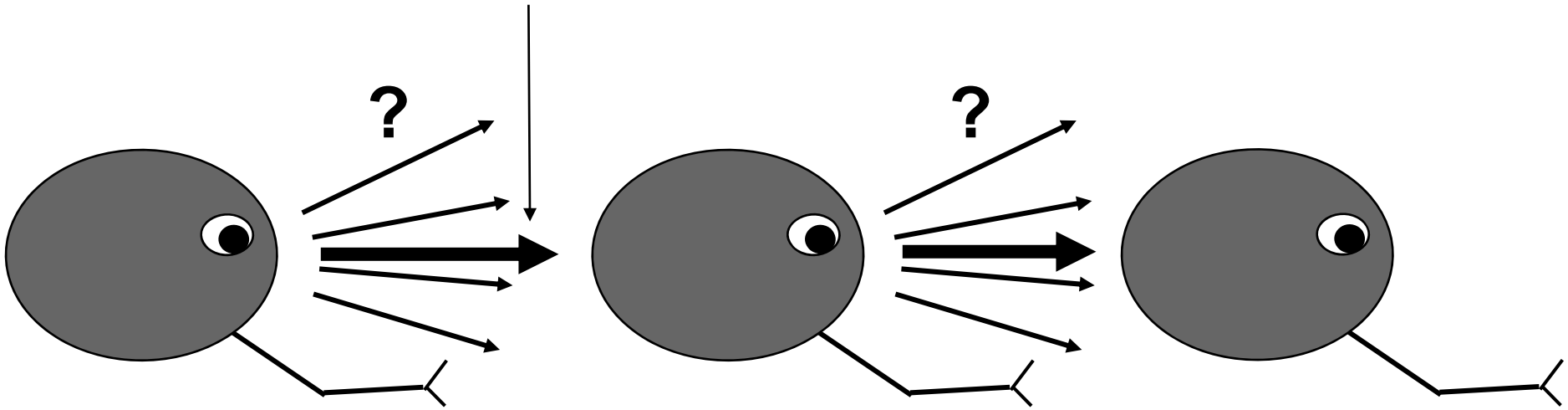
Types of External Environments:

- **Simulators:**
  - ModSAF, JSAF, OneSAF, NGTS, HLA, DIS
- **Games/Game engines:**
  - Unity, Unreal, Quake, Gamebryo, XPlane, EDGE, Full Spectrum Command, Full Spectrum Leader, Starcraft
- **Robotics systems:**
  - ROS, LCM, JAUS, Player/Stage

Integrating Soar with External Environments: Soar Markup Language (SML)

- Domain independent API support all external integration
- Connects to multiple languages: C/C++, Java, Python, TCL
- Supports:
  - single process, across processes, across machines
  - dynamic connection at runtime to multiple clients
  - asynchronous and synchronous operation
  - communication of I/O, run-time and debugging commands
- Integration is fast and straight forward (hours)
- High performance (especially if within process)

Core Soar Function



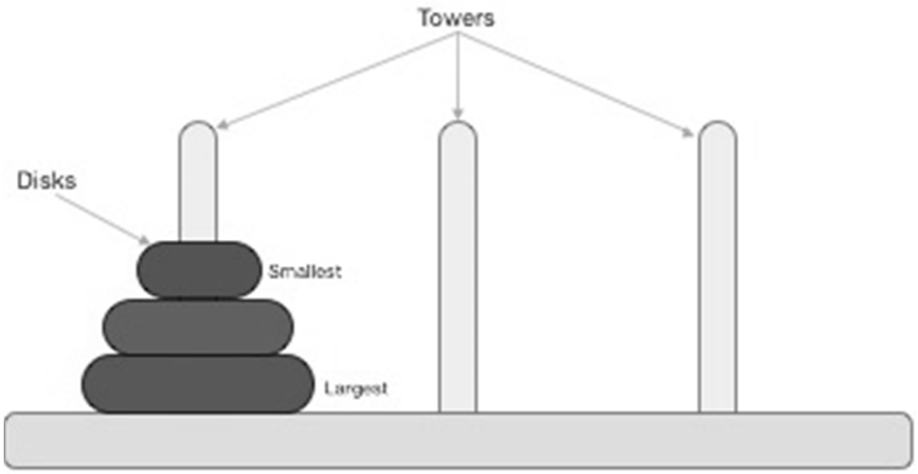
Operator Proposal

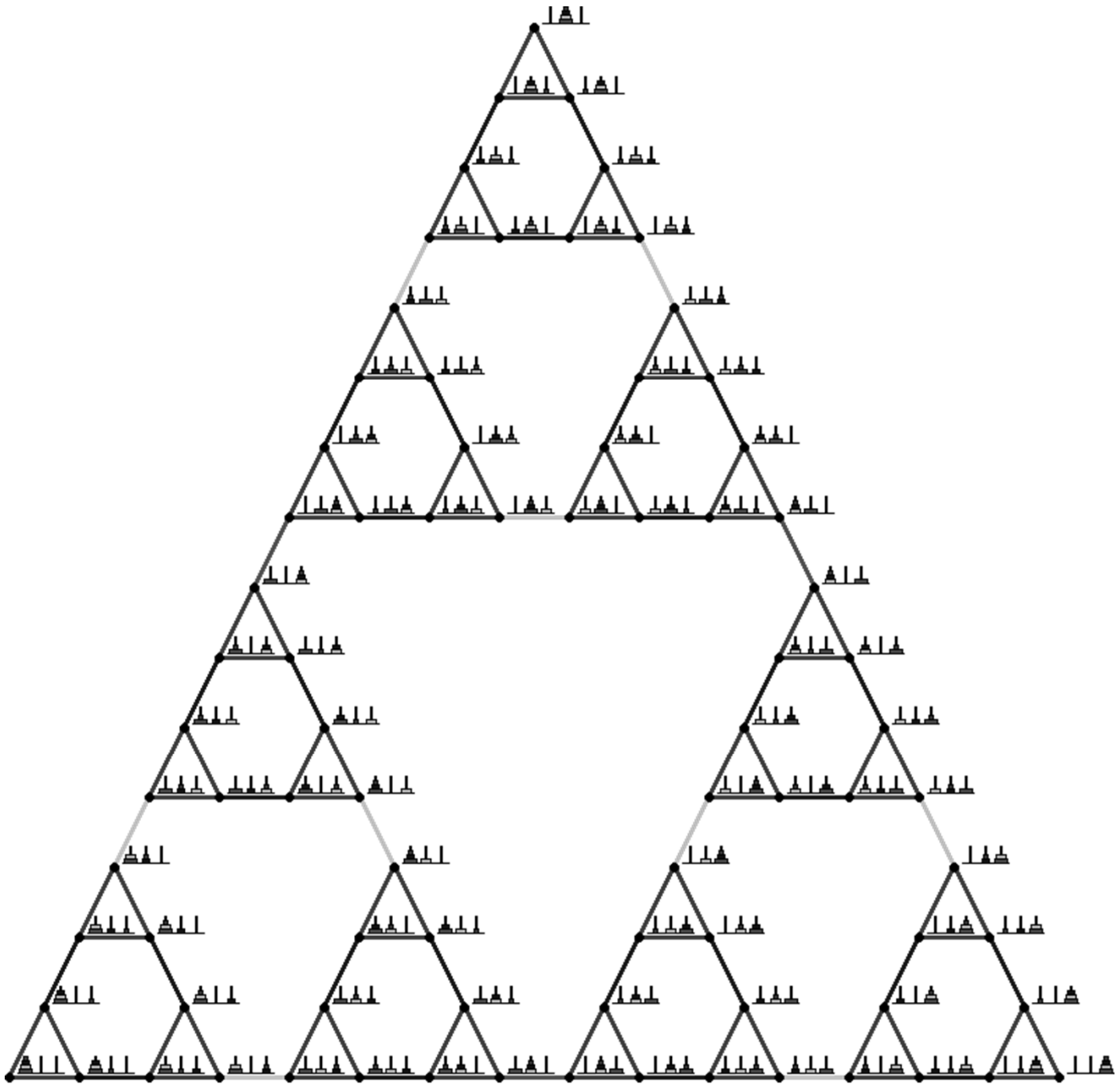
Operator Selection

Operator Application

Problem Spaces

- *State:* the current situation the agent is in
- *Operators:* transition to new state
  - Internal reasoning steps with changes to working memory
    - Logical deduction and inference, simple math, …
  - Retrievals from long-term semantic or episodic memory
  - Mental imagery actions
  - External motor actions
- *Goals:* states to be achieved

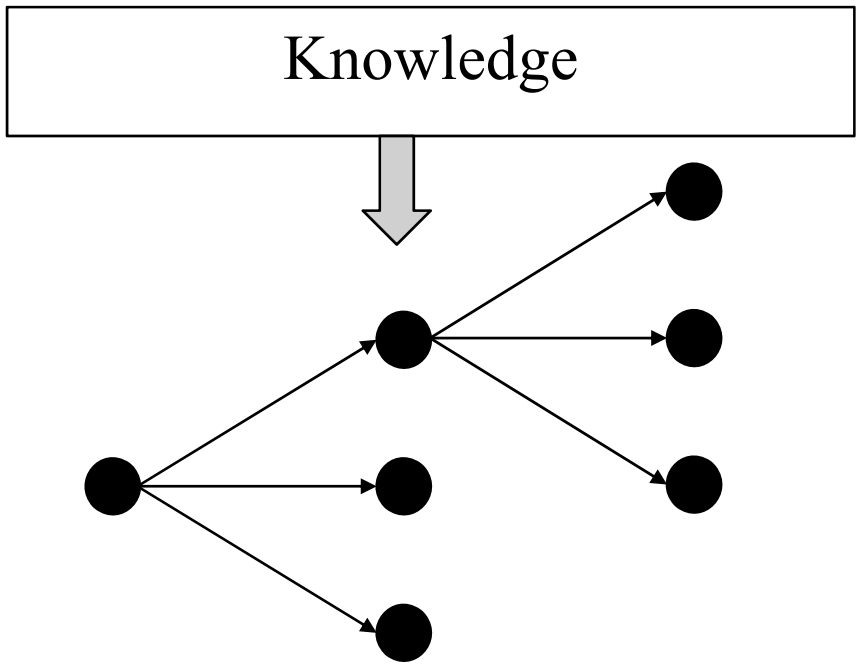




Examples of Using Knowledge

- Counting
- Planning a trip
- Playing Tic Tac Toe
- Robot control
- Natural language processing

Knowledge Search vs. Problem Search

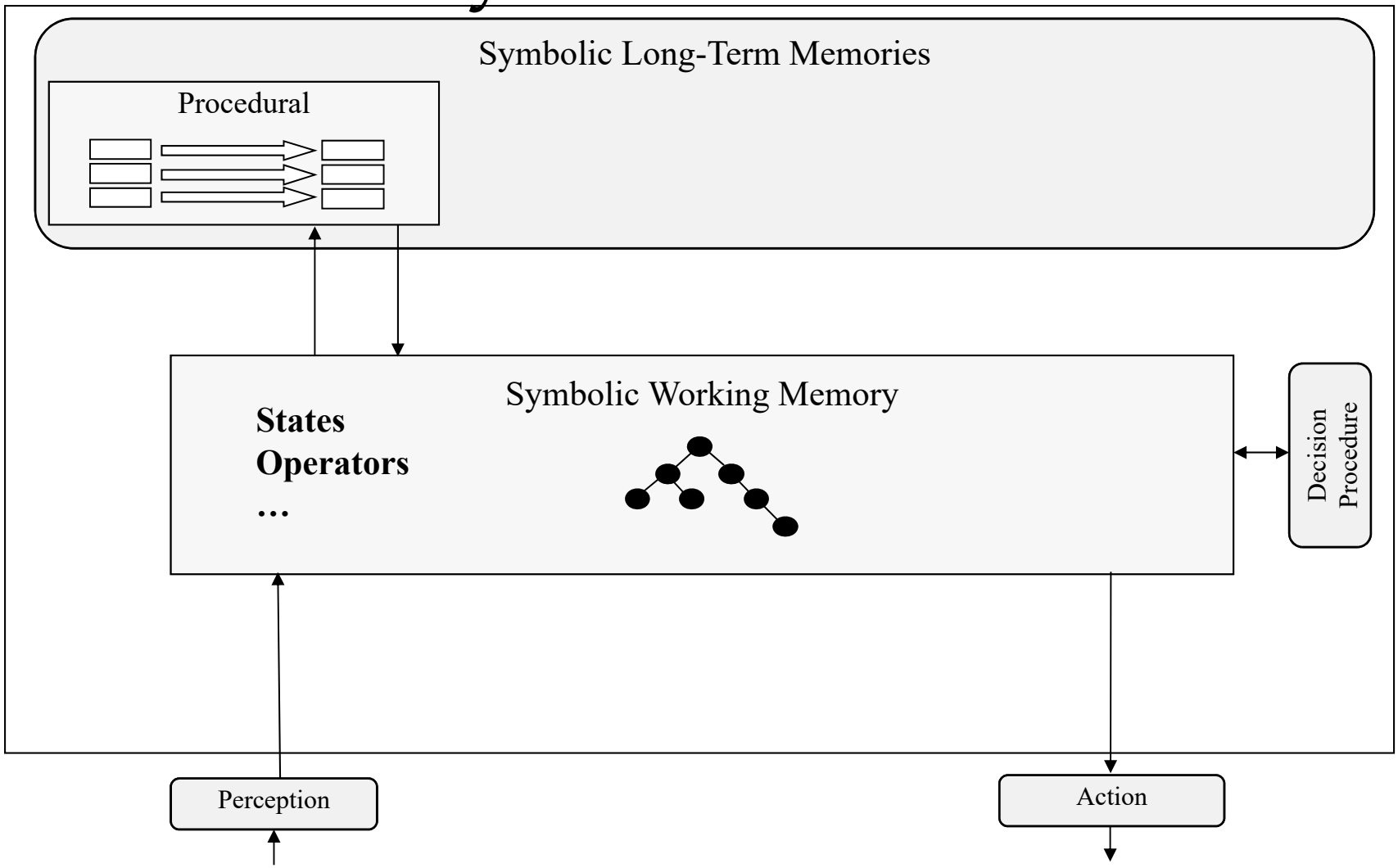


Problem Search
- Generative & combinatorial
- Controlled by knowledge
- Improve with experience

Knowledge Search (match)
- Finding the right rule to fire
- Retrieving something from semantic or episodic memory
- Search is over existing knowledge
- Very fast
- Fixed mechanisms

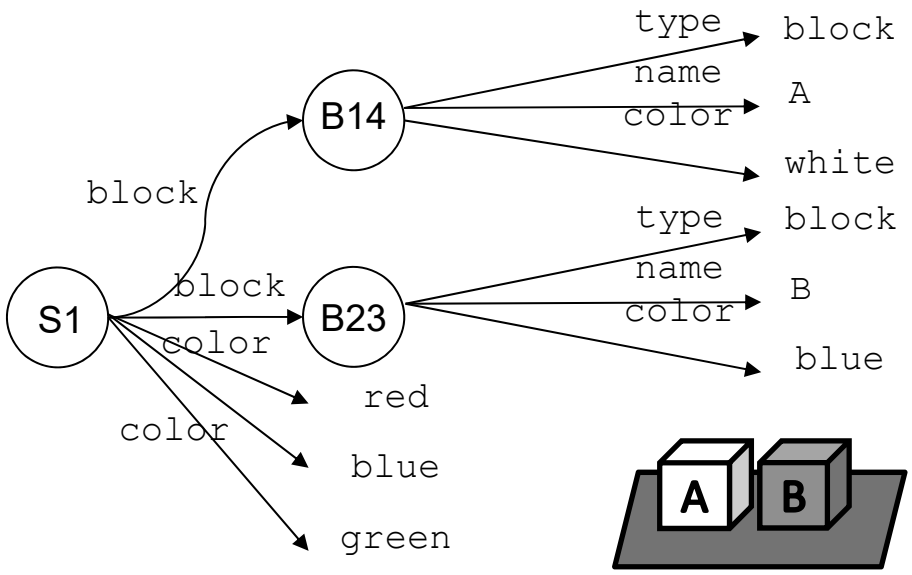
Key idea: Use knowledge search to control problem search

Early Soar Structure



Representation of State in Working Memory

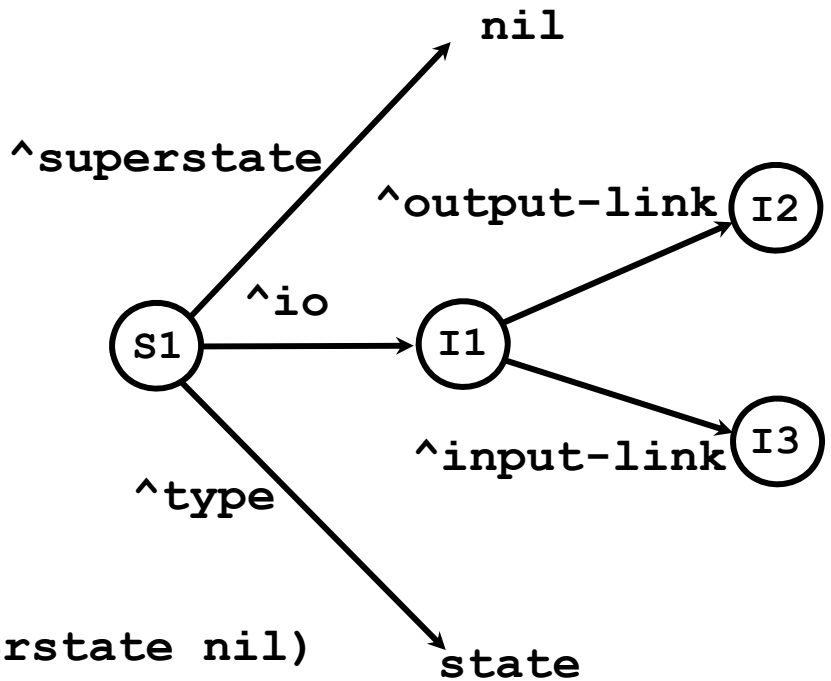
- Working memory consists of individual *elements*.
- Elements are organized as a *graph*.
- Element: (identifier ^attribute value)
  - (s1 ^color red)
- The graph is rooted in the state.




```
(S1 ^block B14)
(S1 ^block B23)
(S1 ^color red)
(S1 ^color blue)
(S1 ^color green)
(B14 ^type block)
(B14 ^name A)
(B14 ^color white)
(B23 ^type block)
(B23 ^name B)
(B23 ^color blue)
```

```
(S1 ^block B14 B23
    ^color red blue green)
(B14 ^type block ^name A
     ^color white)
(B23 ^type block ^name B
     ^color blue)
```

Subset of Initial Working Memory

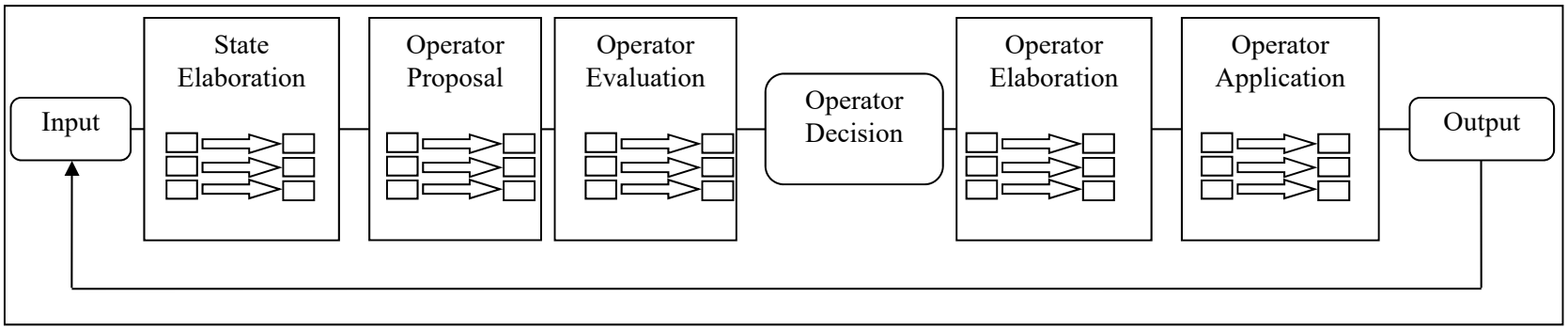



```
(S1 ^superstate nil)
(S1 ^io I1)
(S1 ^type state)
(I1 ^output-link I2)
(I1 ^input-link I3)

(S1 ^io I1 ^superstate nil ^type state)
(I1 ^input-link I3 ^output-link I2)
```

Soar Basic Functions

Soar uses rules to represent procedural knowledge.

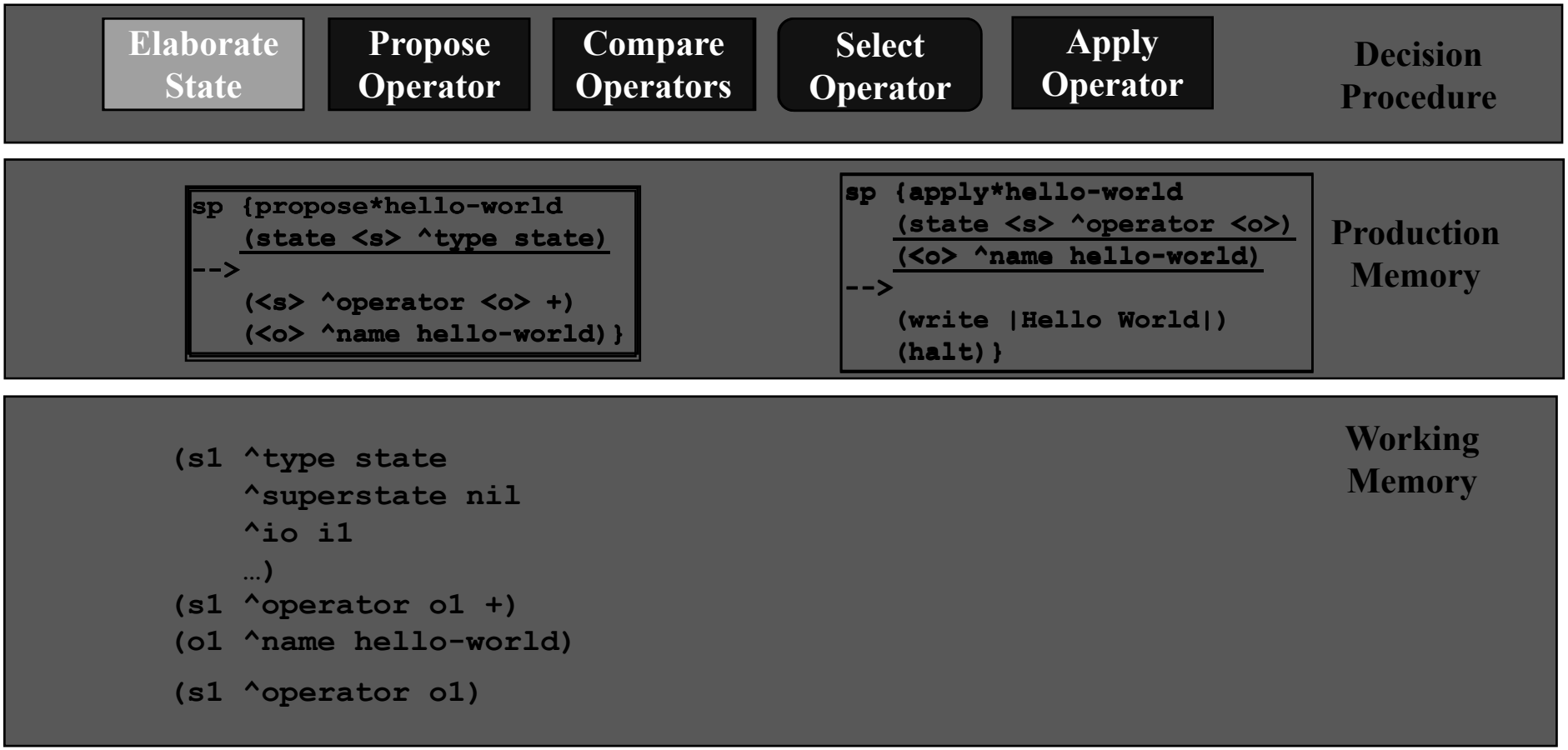
1. Input from environment
2. Elaborate current situation: *parallel rules*
3. Propose and evaluate operators via *preferences*: *parallel rules*
4. Select operator
5. Apply operator: Modify internal data structures: *parallel rules*
6. Output to motor system [and access to long-term memories]



Assumptions:

- Complex behavior arises from multiple cycles.
- Each cycle is bounded processing to maintain reactivity.

Soar 101
Internal Problem Solving

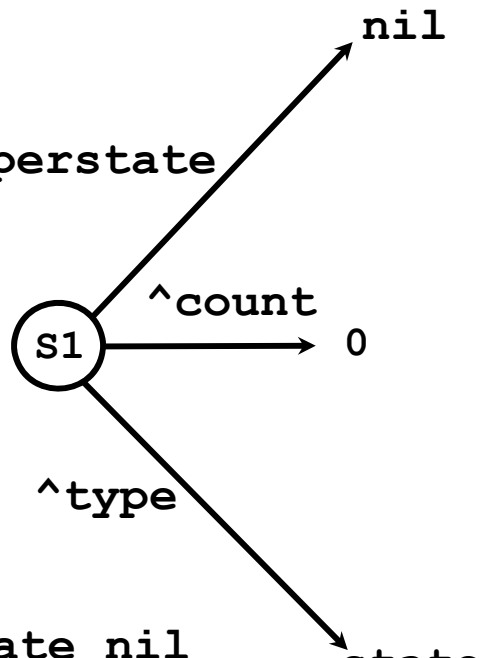


Hello World

Soar Agent to Count to 10

- Operators:
  1. Initialization of count to 0 (s1 ^count 0)
  2. Count: add 1
- State:
  - Current count: (s1 ^count 0)
- Goal detection:
  - Count = 10: (<s> ^count 10)

Count Working Memory




```
(S1 ^superstate nil
    ^count 0
    ^type state)
```

Operator Definition

For every operator, must be defined by at least two rules:

1. Proposal rule creates operator structure in working memory and acceptable preference
2. Application rule tests for selected operator
   - Makes changes to the state
   - Application must change state so proposal no longer matches

Count Operators

Proposal: when operator should be considered

- **propose*initialize-count:**
  - `if the state exists and the state isn't named count, then propose initialize-count.`
- **propose*count:**
  - `if there is a count, then propose count.`

Application: how the operator changes the state

- **apply*initialize-count:**
  - `if initialize-count is selected, then create a count of 0.`
- **apply*count:**
  - `if count is selected, then replace the current-count with (+ 1 current-count).`

if the state exists and the state isn't named count, then propose initialize-count.

```
sp {propose*initialize-count
   (state <s> ^type state)
  -(<s> ^name count)
   -->
   (<s> ^operator <o> +)
   (<o> ^name initialize-count)}
```

if initialize-count is selected, then create a count of 0.

```
sp {apply*initialize-count
   (state <s> ^operator <o>)
   (<o> ^name initialize-count)
   -->
   (<s> ^name count
        ^count 0)}
```

if there is a count, then propose count.

```
sp {propose*count
    (state <s> ^count < 10)
    -->
    (<s> ^operator <o> +)
    (<o> ^name count)}
```

if count is selected, then replace current-count with (+ 1 current-count).

```
sp {apply*count
    (state <s> ^operator <o>
               ^count <count>)
    (<o> ^name count)
    -->
    (<s> ^count <count> -)
    (<s> ^count (+ 1 <count>))}
```

if there is a count, then propose count.

```
sp {propose*count
   (state <s> ^count < 10)
   -->
   (<s> ^operator <o> +)
   (<o> ^name count)}
```

if count is selected, then replace the current-count with (+ 1 current-count).

```
sp {apply*count
   (state <s> ^operator.name count
              ^count <count>)
   -->
   (write (crlf) |Count: | (+ 1 <count>))
   (<s> ^count <count> -
               (+ 1 <count>))}
```

Goal Detection

```
if the count is 10, then halt.
 sp {detect*count10
    (state <s> ^count 10)
    -->
    (halt)}
```

Persistence!

Actions of non-operator application rules ***retract*** when rule no longer matches

- No longer relevant to current situation
- Operator proposals and state elaboration
- Instantiation-support = i-support
- *Rule doesn't test the selected operator and modify state.*
  - Propose operator:
    - `If the current count is less than 10, then propose the count operator.`
  - Elaborate state:
    - `If the count is 1, 3, 5, 7 or 9, then the number is odd.`
  - Create operator preferences:
    - `If playing hearts and not shooting the moon, then avoid taking tricks with hearts or the Queen of Spades.`

Persistence!

Actions of operator application rules *persist* indefinitely

- Otherwise actions retract as soon as operator isn't selected
- Operators perform non-monotonic changes to state
- Operator-support = o-support
- *Rule tests the selected operator and modifies the state*
  - Operator application:
    - **`if operator count is selected and the current-count is <x>, then replace the current-count with (+ 1 <x>).`**

Review of Operators in Working Memory

- To be considered for selection, an operator must have an acceptable preference on the state.

```
(s1 ^operator o1 +)
```

- Operators must have a declarative representation in working memory (something rules can test, such as name).

```
(o1 ^name count)
```

- When an operator is *selected*, there is a working memory element in the state (different than the preference) that was created by the decision procedure.

```
(s1 ^operator o1)
```

- Rules that test for a selected operator, apply the operator by modifying the state.

```
(<s> ^operator <o>)
```

Fibonacci

- Create operators that compute the Fibonacci sequence.
  - 0 1 1 2 3 5 8 13 …
- State representation:
  - Last two numbers
  - count0, count1
  - (<s> ^count0 …
  - ^count1 …)
- Operator updates counts
  - count0 ← count1
  - count1 ← (+ count0 count1)

Fibonacci Operators

- propose*initialize-count:
  - **`if the state exists and there is no count0, then proposal initialize-count.`**
- apply*initialize-count:
  - **`if initialize-count is selected, then create a count0 of 0, count1 of 1.`**

- propose*count-fibonacci:
  - **`if there is a count0, then proposal count-fibonacci.`**
- apply*count-fibonacci:
  - **`if count-fibonacci is selected, then replace the count0 with count1 and replace count1 with (+ count0 count1)`**

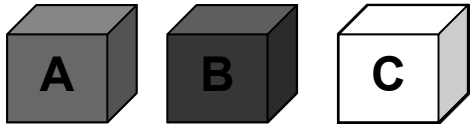
```
sp {propose*initialize-count-fibonacci
    (state <s> ^type state
               -^name fibonacci)
    -->
    (<s> ^operator <o> +)
    (<o> ^name initialize-count-fibonacci)}


sp {apply*initialize-count-fibonacci
    (state <s> ^operator <o>)
    (<o> ^name initialize-count-fibonacci)
    -->
    (<s> ^name fibonacci
         ^count0 0
         ^count1 1)}
```

```
sp {propose*count-fibonacci
    (state <s> ^count0
               ^count1)
    -->
    (<s> ^operator <o> +)
    (<o> ^name count-fibonacci)}

sp {apply*count
    (state <s> ^operator.name count-fibonacci
               ^count0 <c0>
               ^count1 <c1>)
 -->
    (write (crlf) |Next Number: | <c1>)
    (<s> ^count0 <c0> -
                 <c1>
         ^count1 <c1> -
                 (+ <c0> <c1>))}
```

Operators and States for Colored Blocks World

States

- Objects
  - blocks [color, name]
  - paint brushes for specific colors [color]
- Initially all blocks are white



Operators:

- Initialize-blocks-world
- Paint a block with a different paint brush color

Goal:

- All blocks are red

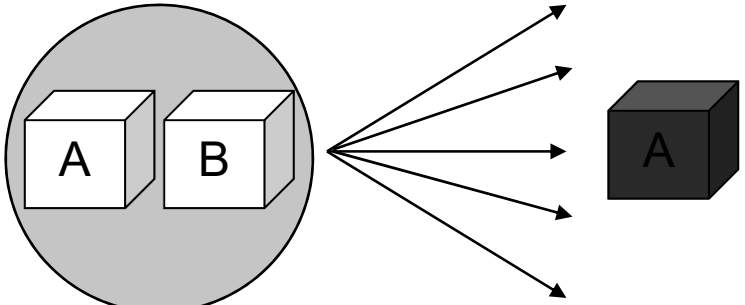
Multiple operators can be proposed at the same time.

Use *preferences* to select between them.

Basic Soar Operation

State	Operators proposed by rules	Proposed operators evaluated by rules	Operator selected by decision procedure	Operator applied by rule

If block [X] is not color [Y]
Then propose Paint [X] with [Y].



O1+. Paint [A] [Red]

O2+. Paint [A] [Blue]

O3+. Paint [A] [Green]

O4+. Paint [B] [Red]

…

If operator has color [Red]
Then make best preference

(O1 >) Best preference

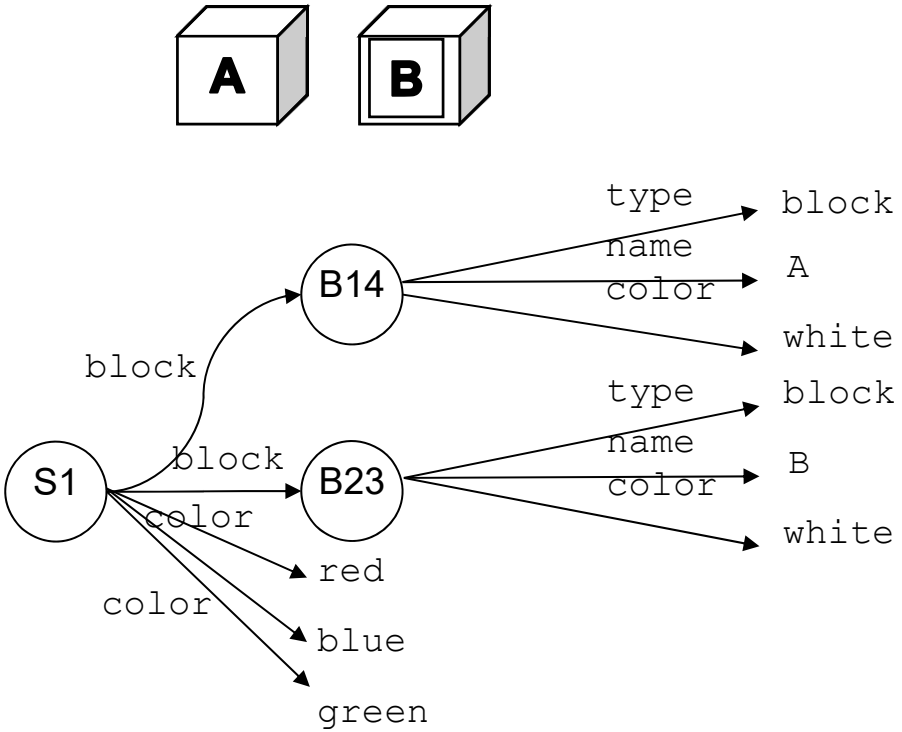
(O4 >) Best preference

Select O1

If operator selected to paint block [X] with color [Y]
Then change color of [X] to [Y].

Example Working Memory

State: blocks [color, name]
paint brushes for specific colors [color]




```
(S1 ^block B14)
(S1 ^block B23)
(S1 ^color red)
(S1 ^color blue)
(S1 ^color green)
(B14 ^type block)
(B14 ^name A)
(B14 ^color white)
(B23 ^type block)
(B23 ^name B)
(B23 ^color white)
```

```
(S1 ^block B14 B23
    ^color red blue green)
(B14 ^type block ^name A
     ^color white)
(B23 ^type block ^name B
     ^color white)
```

Defining the Task in Soar

Create rules for:

- Initialize-color-block operator
  - Propose Initialize-color-block
  - Apply Initialize-color-block
- Color-block operator
  - Propose color-block
  - Apply color-block
  - Select color-block
- Detect goal achieved

If there the top state does not have the name "color-block"
then propose the operator to initialize-color-blocks.

```
sp {propose*initialize-color-blocks
    (state <s> ^type state
               -^name color-block)
    -->
    (<s> ^operator <o> +)
    (<o> ^name initialize-color-blocks)}
```

If the initialize-color-blocks operator is selected,
then add the name to the state and add the colors, and create the blocks A, B, and C.

```
sp {apply*initialize-color-blocks
    (state <s> ^operator.name initialize-color-blocks)
    -->
    (<s> ^name color-block
         ^color red green blue
         ^block <b1> <b2> <b3>)
    (<b1> ^type block
          ^color white
          ^name A)
    (<b2> ^type block
          ^color white
          ^name B)
    (<b3> ^type block
          ^color white
          ^name C)}
```

```
If there is a block that has a color different than an existing color,
   then propose the operator to color that block that color,
   also create an indifferent preference.

sp {propose*color-block
    (state <s> ^color <color>
               ^block <block>)
    (<block> ^color <> <color>)
    -->
    (<s> ^operator <o> +)
    (<s> ^operator <o> =)
    (<o> ^name color-block
         ^color <color>
         ^block <block>)}
```

(<s> ^operator <o> + =)

```
If there is an operator selected to color a block a color,
         color that block that color.

sp {apply*color-block
   (state <s> ^operator <o>)
   (<o> ^name color-block
        ^color <color>
        ^block <block>)
   (<block> ^name <name>
            ^color <old-color>)
 -->
   (write (crlf) |Paint block | <name> | | <color>)
   (<block> ^color <old-color> -
            ^color <color>)}
```

```
sp {prefer*color-red
    (state <s> ^operator <o> +)
    (<o> ^color red)
    -->
    (<s> ^operator <o> > =)}

sp {prefer*avoid*color-green
    (state <s> ^operator <o> +)
    (<o> ^color << green blue >>)
    -->
    (<s> ^operator <o> <)}

# alternative
sp {prefer*color-red-to-blue
    (state <s> ^operator <o1> +
               ^operator <o2> +)
    (<o1> ^color red)
    (<o2> ^color blue)
    -->
    (<s> ^operator <o1> > <o2>)}
```

Goal Detection

```
If the three blocks are color "red" then halt.

sp {detect*color-red
    (state <s> ^block <a> <b> <c>)
    (<a> ^name A ^color red)
    (<b> ^name B ^color red)
    (<c> ^name C ^color red)
    -->
    (halt)}
```

More Complex Operator Application

- Count the number of times an operator applied.
- Modify initialization rule so includes ^count 0 for each color.
- Add application rule that increments count in parallel with painting:

```
sp {apply*color-block*count
    (state <s> ^operator <o>)
    (<o> ^name color-block
         ^block <block>)
    (<block> ^count <count>)
    -->
    (<block> ^count <count> -
             ^count (+ 1 <count>))}
```

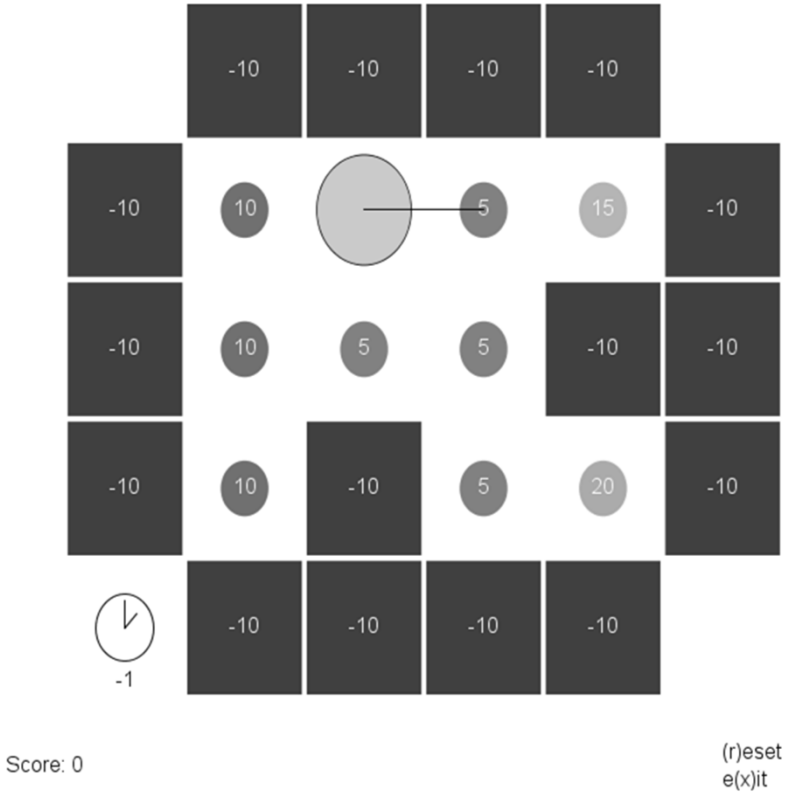
- Change goal to print count.

What you don't do in Soar productions

1. No replace command
   - Must explicitly add and remove structures in WM
2. Complex calculations should be done via I/O or SVS
   - External computational aids (calculators, …)
3. Can't do math in conditions
   - Conditions can only test existence or absence of WME's
   - Equality or inequality of identifiers and constants
     - Simple inequality of numbers (>, <, >=, <=, <>)
4. Only simple calculations in actions
5. Cannot match variables in actions

Break

Simple Eater



Actions:

- forward: move one cell
- rotate: turn right

State:

- sensory data: input-link
- internally maintained: state

Get points for eating food.

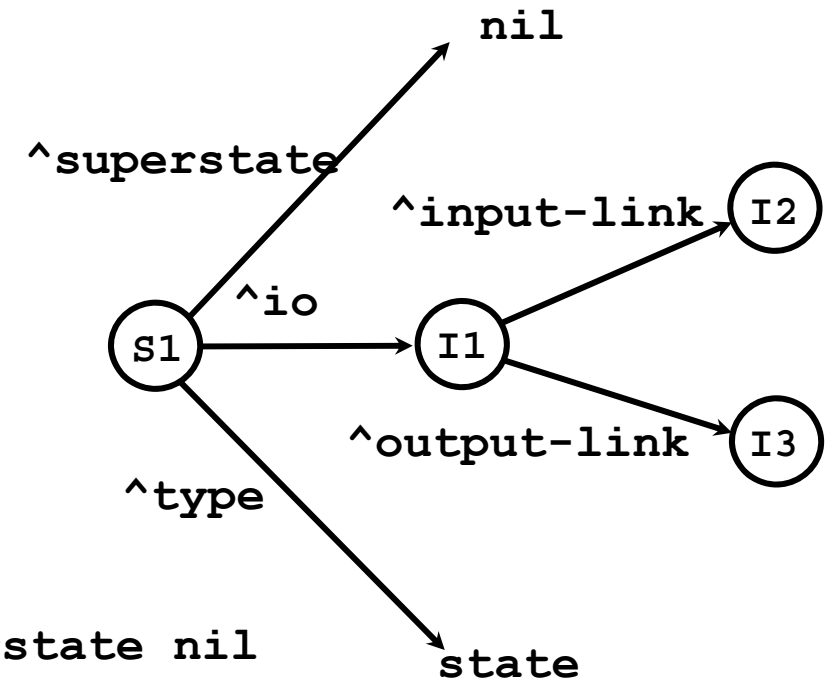
-1 for each forward/rotate.

Input/Output in Soar

- All input and output happens through working memory.
- Input is added by perception during input phase:
  - **`(<s> ^io.input-link <input>)`**

- Output commands are created by rules on:
  - **`(<s> ^io.output-link <output>)`**
  - Sent to motor system in output phase

Subset of Initial Working Memory




```
S1 ^superstate nil
S1 ^io I1
S1 ^type state
I1 ^output-link I2
I1 ^input-link I3

(S1 ^io I1 ^superstate nil ^type state)
(I1 ^input-link I3 ^output-link I2)
```

Propose and apply initialize-random

If there the top state does not have the name "eater" then propose the operator to initialize-eater.

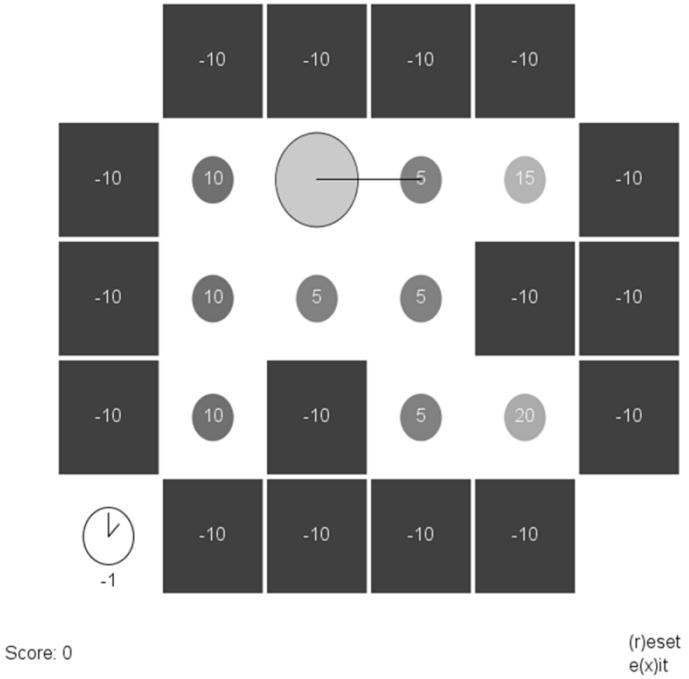
```
sp {propose*initialize-eater
    (state <s> ^type state
               -^name eater)
    -->
    (<s> ^operator <o> +)
    (<o> ^name initialize-eater)}
```

```
sp {apply*initialize-eater
    (state <s> ^operator.name initialize-eater)
    -->
    (<s> ^name eater)}
```

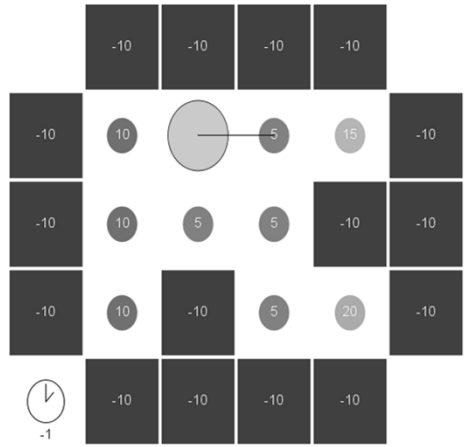
Included in the base agent – you don't need to write these rules.

"Blinking"

- When does a value change on the input-link?
- Why is that important?

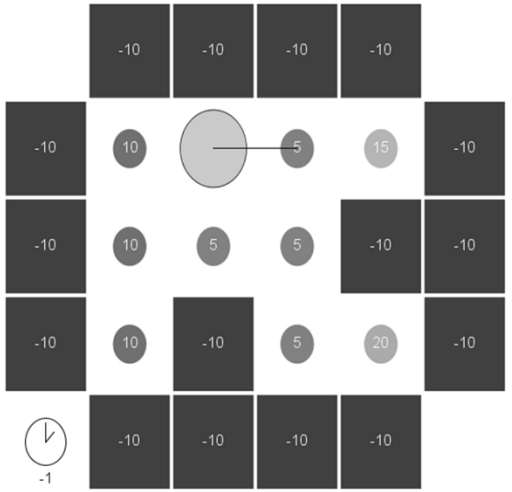


Simple Eater Input-link



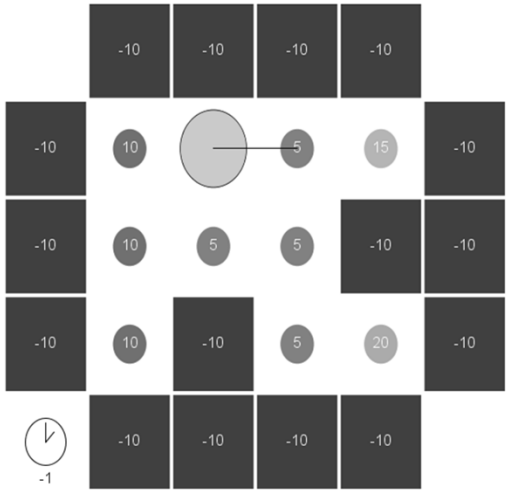

```
The input-link maintains all sensory data.

(<s> ^io.input-link <input>)
(<input> ^east red            # absolute directions and contents
         ^north wall
         ^south red           # these change with forward
         ^west purple

         ^back purple         # relative directions and contents
         ^front red
         ^left wall           # these change with rotate or forward
         ^right red

         ^orientation east    # this changes with rotate

         ^score 0
         ^score-diff 0
         ^food-remaining 10 # 0 when eaten all food

         ^x 1                 # these change with forward
         ^y 2

         ^time 1              # this changes with rotate/forward)
```

Propose and Apply Forward




```
sp {random*propose*forward
   (state <s> ^name eater
              ^io.input-link.front)    # will blink
-->
   (<s> ^operator <op> + =)
   (<op> ^name forward)}


sp {apply*forward
   (state <s> ^operator <op>
              ^io.output-link <out>)
   (<op> ^name forward)
-->
   (<out> ^forward <f>)}
```

Propose and Apply Rotate




```
sp {random*propose*rotate
   (state <s> ^name eater
              ^io.input-link.front) # will blink
-->
   (<s> ^operator <op> + =)
   (<op> ^name rotate)}


sp {apply*rotate
   (state <s> ^operator <op>
              ^io.output-link <out>)
   (<op> ^name rotate)
-->
   (<out> ^rotate <r>)}
```

Cleaning up output-link

Need to remove structures on the output-link.
Can do this when an operator is selected and get back ^status complete.

```
sp {apply*cleanup*output-link
   (state <s> ^operator <op>
              ^io.output-link <out>)
   (<out> ^<cmd> <id>)
   (<id> ^status complete)
-->
   (<out> ^<cmd> <id> -)
}
```

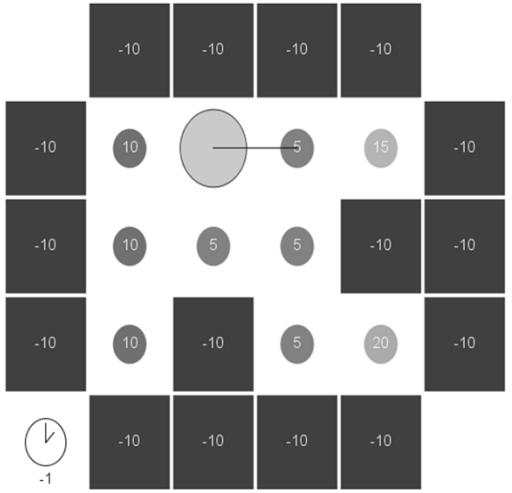
Included in the base agent – you don't need to write this.

Detecting Completion

If the task is complete, halt.

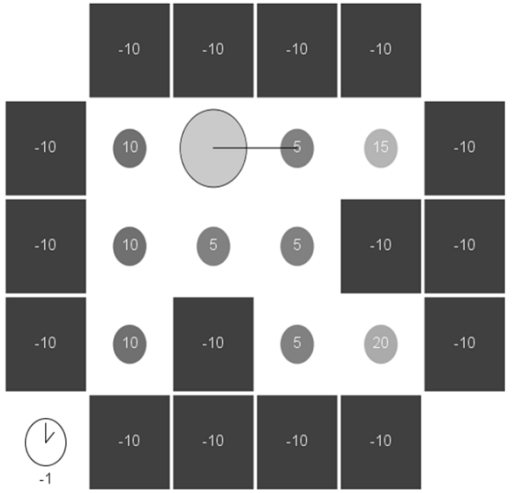
```
sp {task*complete
    (state <s> ^name eater
               ^io.input-link.food-remaining 0)
-->
    (halt)
}
```

Smarter Eater



- Reject moving forward into walls
- Avoid moving forward into empty cells

Reject wall, Avoid empty




```
sp {eater*reject*forward*wall
   (state <s> ^operator <o> +
              ^io.input-link.front wall)
   (<o> ^name forward)
-->
   (<s> ^operator <o> -)}


sp {eater*avoid*forward*empty
   (state <s> ^operator <o> +
              ^io.input-link <input>)
   (<input> ^<< back left right >> { <> empty <> wall }
            ^front empty)
   (<o> ^name forward)
-->
   (<s> ^operator <o> <)}
```