Soar-RL Tutorial

Soar Workshop 29

Nate Derbinsky

University of Michigan

Setting Expectations

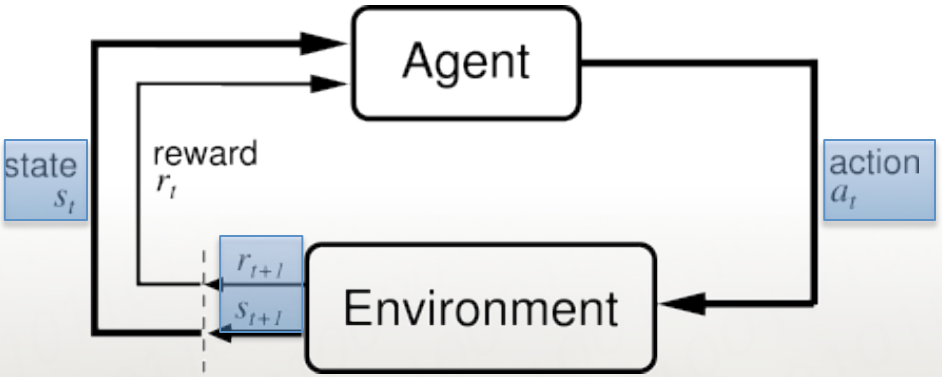
- This is **not** a tutorial on *reinforcement learning*
  - Reinforcement Learning: An Introduction (Richard S. Sutton, Andrew G. Barto)

- Topics
  - Soar-RL as a learning mechanism
  - Agent design
  - Architectural details
  - Useful commands

Some History

- 2004
  - Initial implementation
  - *Soar-RL: Integrating Reinforcement Learning with Soar*
    - Shelley Nason, John Laird (ICCM)
- 2007
  - *The Importance of Action History in Decision Making and Reinforcement Learning*
    - YJ Wang, John Laird (ICCM)
- 2008
  - Re-engineered Soar-RL released as 8.6.4-beta, then 9.0.0
  - *A Computational Unification of Cognitive Control, Emotion, and Learning*
    - Bob Mariner (Dissertation)
- 2009
  - Soar-RL refinements (9.0.1)
  - *Learning to Use Episodic Memory*
    - Nick Gorski, John Laird (ICCM)

Some RL Terminology



- Agent's goal: maximize total amount of reward received over the long run
- Policy: mapping from states to probabilities of selecting each possible action
- RL specifies how the agent changes its policy as a result of its experience

Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction

Numeric Indifferent Preferences

- `(<state> ^operator <op> = number)`
  - **number**, the value of the preference, is a numeric constant

- The value of the numeric indifferent preference may bias selection of the operator from amongst indifferent preferences

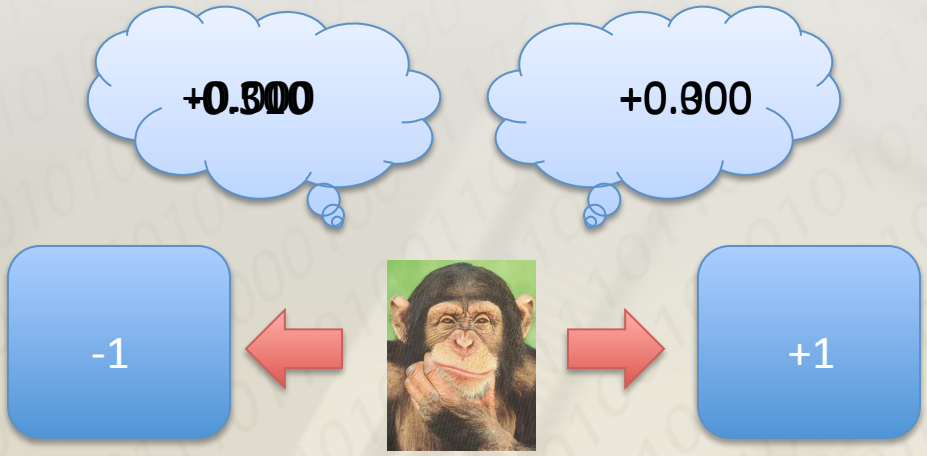
RL as a Learning Mechanism

- Soar-RL directly modifies numeric indifferent preference values such as to maximize the expected receipt of future reward

- By modifying preference values in procedural memory, Soar-RL alters the outcome of operator selection, thereby affecting agent behavior

Left-Right Agent Demo

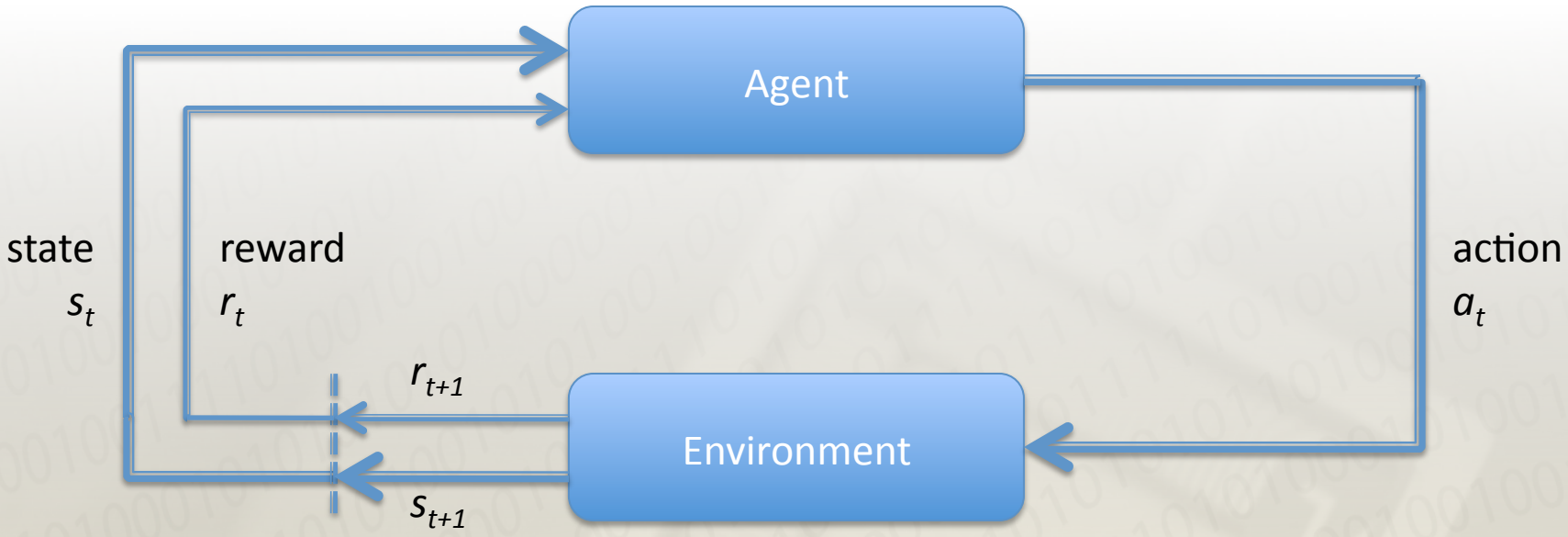
- Load Debugger
- source “SoarLibrary/Demos/left-right.soar”
- srand 5041229 (SOAR29)
- p –rl, run, init; p –rl, run, init; p –rl, run, init; p –rl



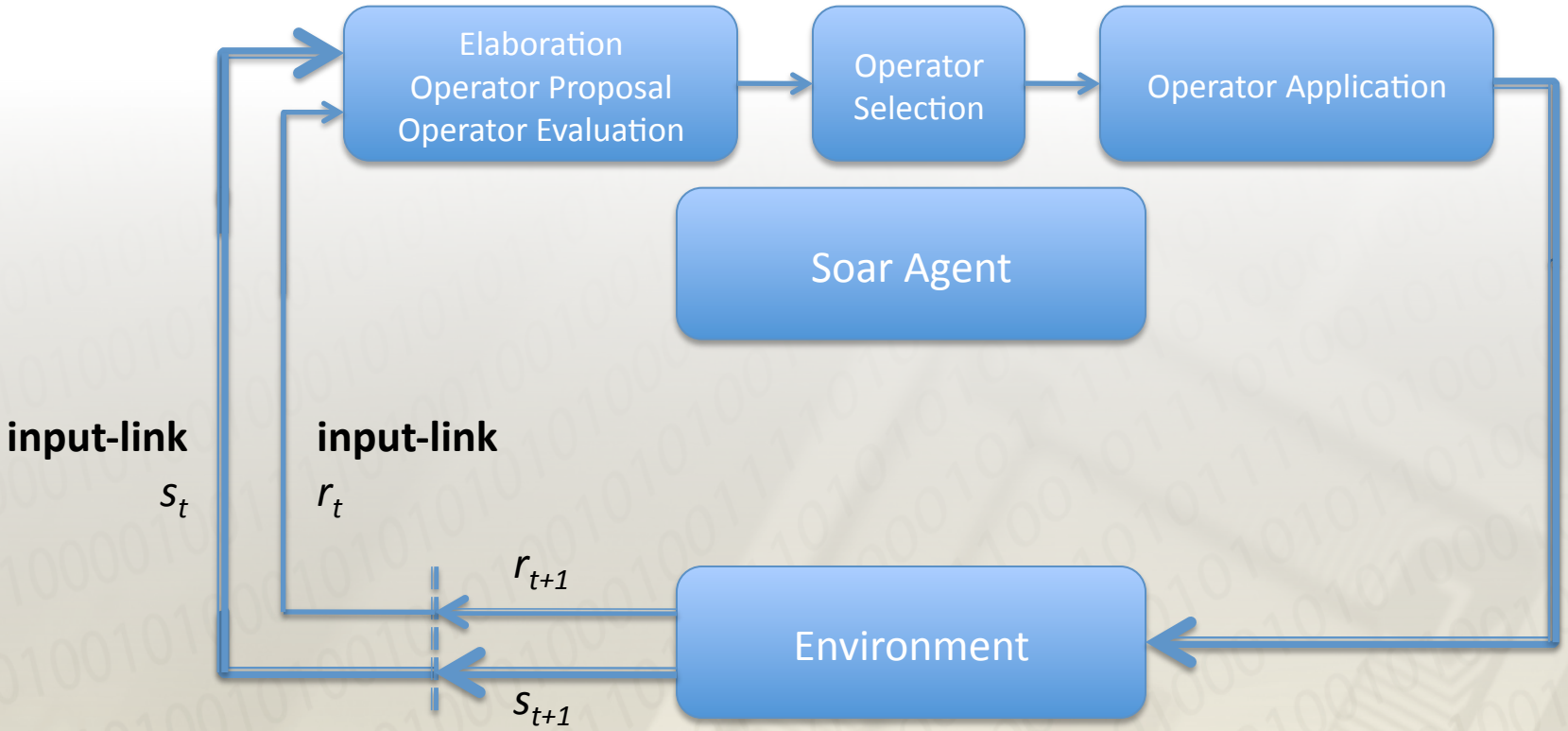
Soar-RL Sequence

Time	Input	Propose	Decide	Apply	Output
t	$state_t$				
t+1					

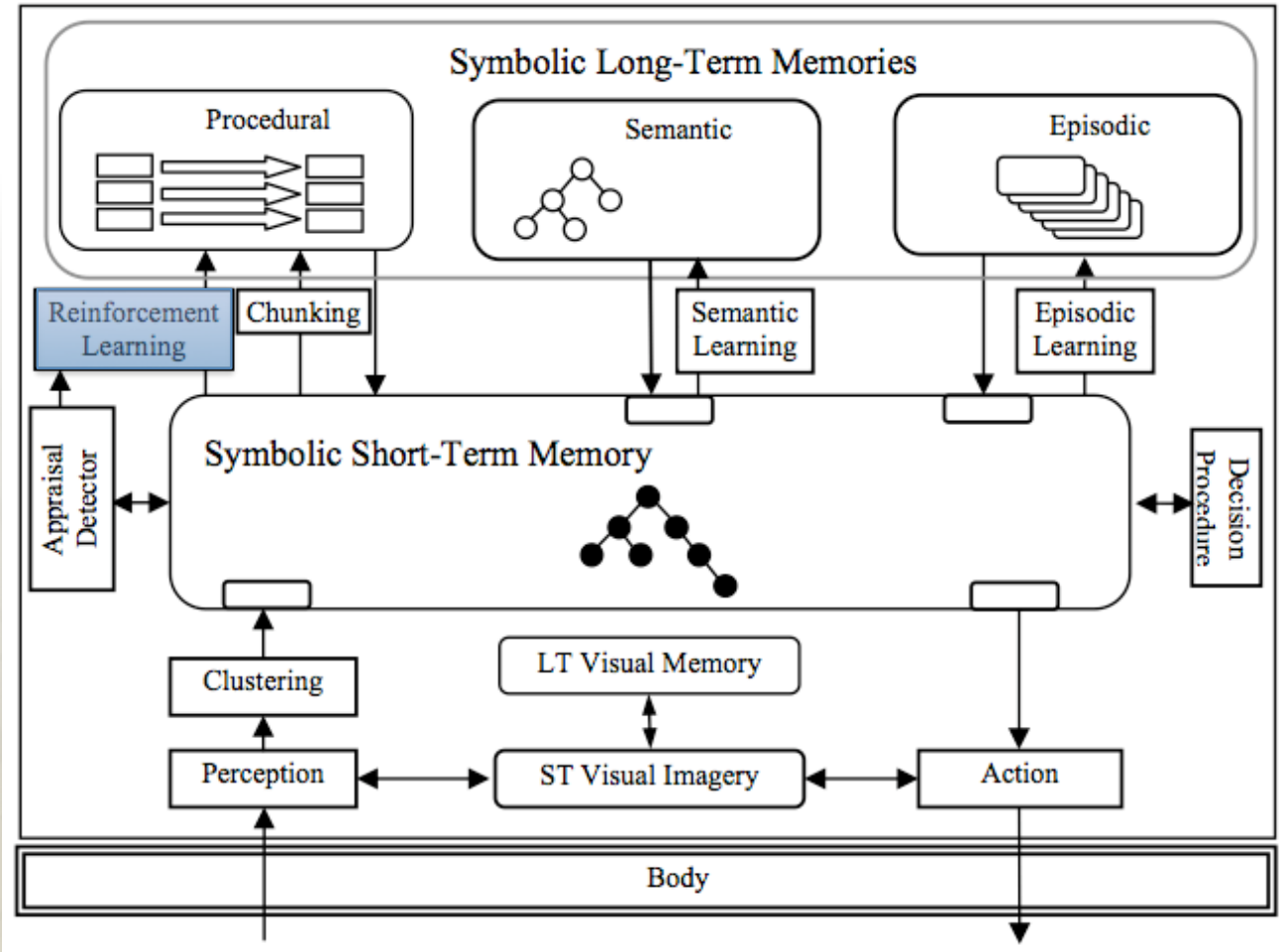
RL Agent-Environment Interface



Soar-RL Agent-Environment Interface



Soar-RL within Soar 9



Soar-RL Agent Design

- To take advantage of the Soar-RL architectural mechanism, an agent must implement two components:
  - Soar-RL compatible preferences
  - Reward rules

Soar-RL Rules

- Operator preferences that are recognized as updateable by Soar-RL must be proposed in a special form:
  - LHS can be anything
  - RHS must be a single numeric indifferent preference with a constant value

```
sp {my*rl*rule
   (state <s> ^operator <op> +
              ^condition-a alpha
              ^condition-b beta)
   (<op> ^name my-op)
-->
   (<s> ^operator <op> = 2.3)
}
```

Soar-RL Rule Example

```
sp {left-right*rl*left
    (state <s> ^name left-right
               ^operator <op> +)
    (<op> ^name move
          ^dir left)
-->
    (<s> ^operator <op> = 0)
}
```

Reward

- Upon creation of a new state within working memory, the architecture will automatically create a **reward-link** structure

- At the beginning of each decision phase, the architecture will collect all properly located numeric constants (integer or float) on each state's **reward-link**:
  - `state ^reward-link.reward.value *`

- When performing an update for a state, the architecture will consider all reward collected at that state's **reward-link**

- The **reward-link** is not part of the **io-link** and is not modified directly by the environment

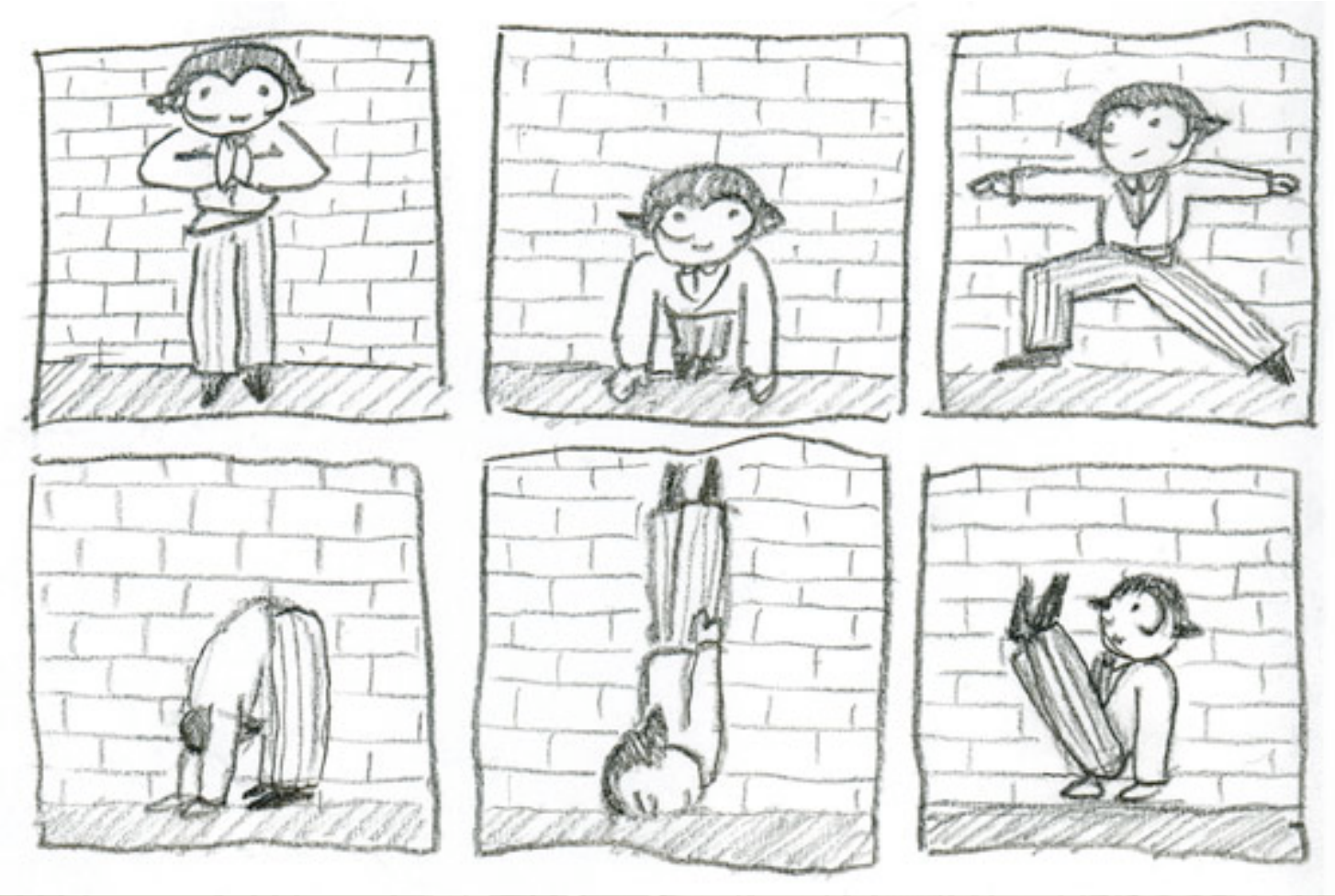
Reward Rule Example

```
sp {left-right*reward*left
    (state <s> ^name left-right
               ^location left
               ^reward-link <r>)
-->
    (<r> ^reward.value -1)
}
```

Left-Right Agent

- Load “SoarLibrary/Demos/left-right.vsa” in VisualSoar

2-Minute Break



Water Jug RL

- Demo
- Soar-RL compatible preferences
- Reward rule

Water Jug RL Demo

- Load Debugger
- Water Jug RL button
- run, init, run, init, run, init

Water Jug: RL Preferences

- Step 1
  - Modify existing proposal rules (propose*empty, propose*fill, propose*pour) such that they no longer propose indifferents (i.e. remove the "=")

- Step 2
  - Create Soar-RL rules to represent the action policy for empty, pour, and fill in all configurations of jugs
    - 3 * 2 * 4 * 6 = 144 rules!!!

Complex Policies

- In order for Soar-RL to affect selection of an operator in a particular state, a Soar-RL rule must exist to represent the state-operator pair

- With complex agents, the requirement of manually generating these rules is unreasonable
  - Solutions: scripting, **gp**, or **templates**

The gp Command

- The **gp** command defines a pattern used to generate and source a set of Soar productions
  - **`gp {production body}`**
- Patterns are whitespace-separated values in square brackets; every combination across all square-bracketed value lists will be generated
- Pros:
  - very fast (all computation done at source)
- Cons
  - limited expressability (can create unnecessary rules)
  - all values must be known at design time

gp Example

```
gp {water-jug*fill
    (state <s> ^name water-jug
               ^operator <op> +
               ^jug <j1> <j2>)
    (<op> ^name fill ^fill-jug.volume [3 5])
    (<j1> ^volume 3 ^contents [0 1 2 3])
    (<j2> ^volume 5 ^contents [0 1 2 3 4 5])
-->
    (<s> ^operator <op> = 0)
}
```

Soar-RL Templates

- Template have variables that are filled in to generate Soar-RL rules as they are encountered

- A rule is a template rule if
  - It has a **:template** flag
  - Adheres to the format of a Soar-RL rule
    - Can use a variable as numeric indifferent value

- Pros
  - only creates rules as they are encountered
- Cons
  - VERY slow during run-time

Template Example

```
sp {water-jug*fill
   :template
   (state <s> ^name water-jug
              ^operator <op> +
              ^jug <j1> <j2>)
   (<op> ^name fill ^fill-jug.volume <vol>)
   (<j1> ^volume 3 ^contents <small-c>)
   (<j2> ^volume 5 ^contents <large-c>)
-->
   (<s> ^operator <op> = 0)
}
```

Water Jug RL Agent

- Implement gp commands for fill, empty, pour
- Add reward to **water-jug*detect*goal*achieved** rule

2-Minute Break

Next Up: Architectural Details

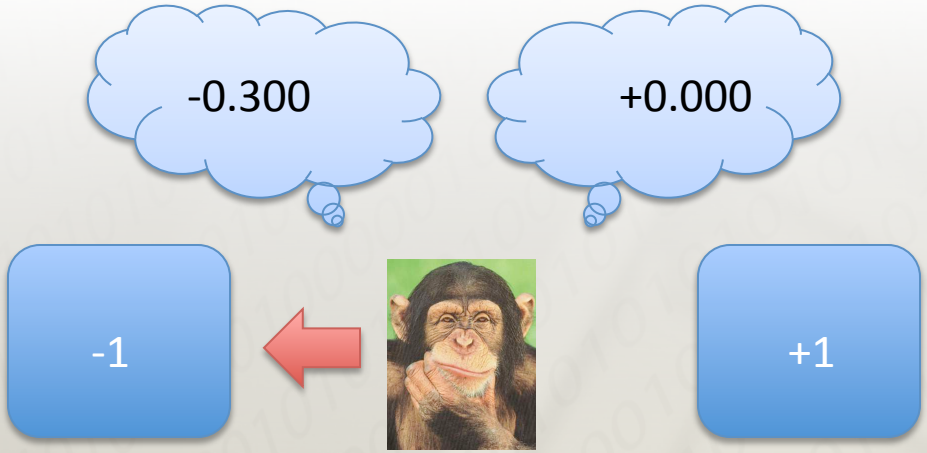
Numeric and Symbolic Preferences

- Symbolic preferences take precedence over numeric preferences
- Symbolic preferences are processed first, and only if there are tied operators remaining are numeric preferences examined
- Example
  - O1 > O2
  - O1 = 0
  - O2 = 2.1

O1 is Selected

Exploration vs. Exploitation

- Recall:



- Why didn't the agent choose **right**?

Exploration Policies

- For reinforcement learning to discover the optimal policy, it is necessary that the agent sometimes choose an action that does not have the maximum predicted value

- This exploration policy is set using the **indifferent-selection** command
  - `indifferent-selection <policy>`

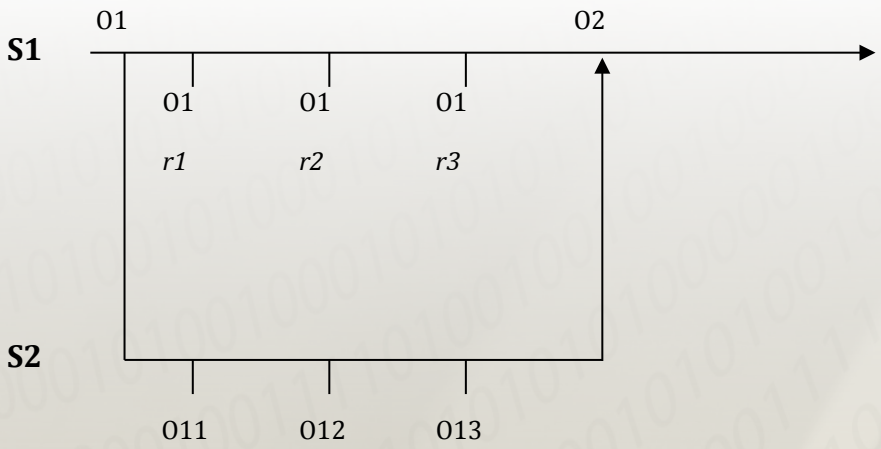
- Policies: boltzmann, epsilon-greedy (default), softmax, first/last (deterministic)

Gaps in Rule Coverage

- Gap
  - one or more contiguous decision cycles during which no Soar-RL rules fire

- By default, Soar-RL will automatically propagate RL updates over gaps, discounted with respect to the length of the gap (defined as the number of decision cycles)

RL in Sub-Goals



- Rewards at S1 after O1 are attributed to O1, discounted with respect to the number of decision cycles
- Rewards at S2 are attributed to the respective operator
- After O13, reward is checked at S2 and, if present, attributed directly to O13

Useful Commands

- Manipulating Soar-RL parameters
- Trace feedback
- Print extension
- Excise extension
- Other commands

Soar-RL Parameters

- Get a parameter
  - **`rl [-g|--get] <name>`**
- Set a parameter
  - **`rl [-s|--set] <name> <value>`**
- Get all values
  - **`rl`**

- Soar-RL is **disabled** by default, to enable:
  - **`rl --set learning on`**

Soar-RL Trace Information

- **`watch --rl`**
  - Provides useful information about gaps and numeric indifferent updates

Soar-RL print Extension

`print [-r|--rl]`

- Prints all Soar-RL rules with the number of updates and current numeric indifferent value of each

```
>print --rl
left-right*rl*right  1.  0.3
left-right*rl*left  2.  -0.51
```

- Common for saving Soar-RL rules at the end of a run

```
command-to-file /path/to/myfile print --full --rl
```

Soar-RL excise Extensions

- **`excise [-r|--rl]`**
  - Removes all Soar-RL rules (including those created from templates)

- **`excise [-T|--template]`**
  - Removes all Soar-RL templates

Other Commands

- **`predict`**
  - Determines, based upon current operator proposals, which operator will be chosen during the next decision phase
- **`select <id>`**
  - Forces the selection of an operator, whose **id** is supplied as an argument, during the next decision phase
- **preferences**

Additional Resources

- Soar-RL Manual and Tutorial
  - In the “Documentation” directory of all releases
- Soar-RL Demo Agents
  - In the “SoarLibrary/Demos” directory of all releases
  - *left-right*: basic tutorial agent
  - *rl-unit*: demonstrates update behavior over gaps/sub-goals
- Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
  - http://www.cs.ualberta.ca/%7Esutton/book/ebook/the-book.html