Soar-RL Tutorial

Soar Workshop 30

Nate Derbinsky

University of Michigan

May 19, 2010

Setting Expectations

This is **not** a tutorial on RL

Reinforcement Learning: An Introduction

Richard S. Sutton, Andrew G. Barto

Topics

- Soar-RL as a learning mechanism
- Agent design
- Additional resources

Procedural Learning Mechanisms

Chunking

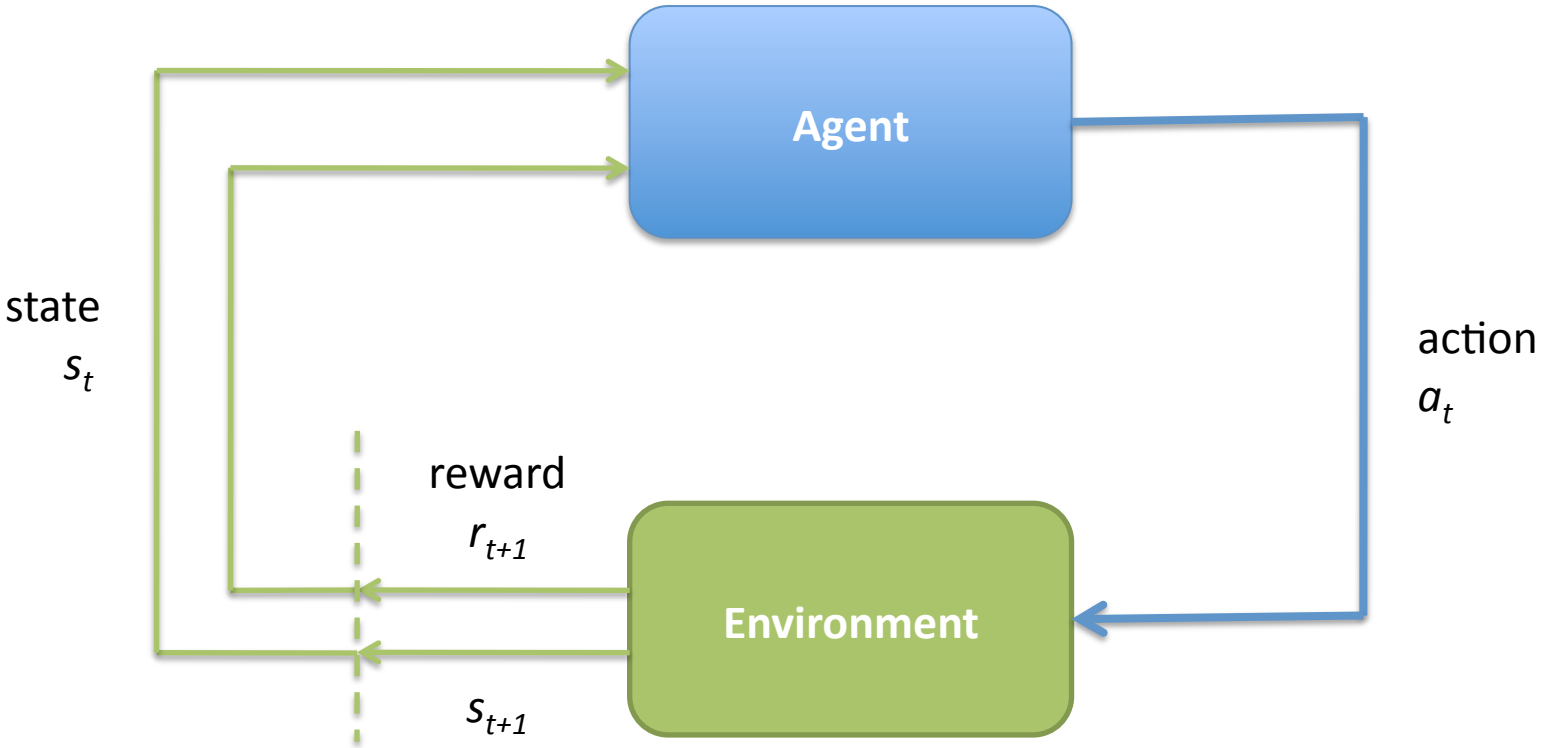
- Converts *deliberation* in substates into *reaction* via rule compilation
- Creates new rules

Reinforcement Learning

- *Tunes* operator numeric preferences to reflect expectation of action reward
- Updates actions of existing rules

Reinforcement Learning Cycle

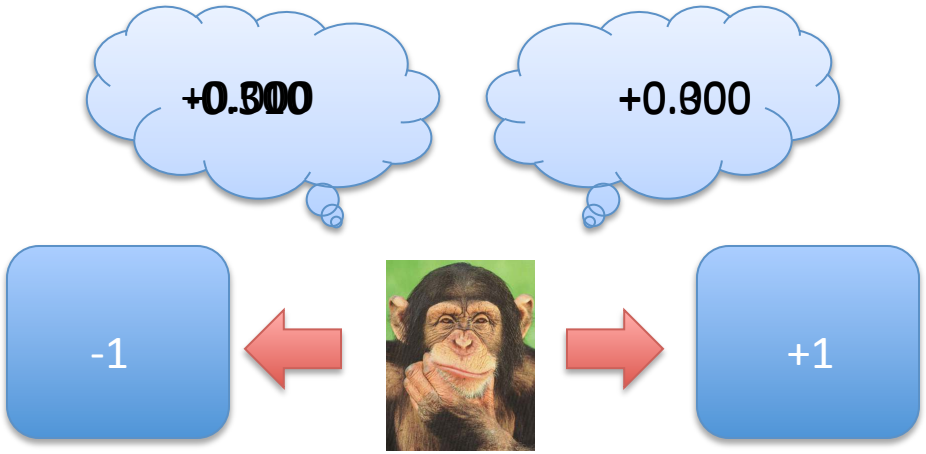
Goal: action selection policy such as to maximize
expected receipt of future reward



Demo: Left-Right Agent

Soar Java Debugger

- source $SOAR_HOME/share/soar/Demos/left-right.soar
- srand 5041229 (SOAR29), watch --rl
- p --rl, run, init; p --rl, run, init; p--rl, run, init; p --rl



Mapping RL to Soar

- Action Selection -> Numeric Indifferent Preferences

  `(<state> ^operator <op> = number)`

  – May bias selection of the operator amongst indifferent preferences

- Reward Signal -> Reward-Link

  Special location in working memory to collect reward signal

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

Soar-RL Sequence

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

Soar-RL Sequence

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

Soar-RL Sequence

Time	Input	Propose	Decide	Apply	Output
t	$state_t$	evaluate $operators_t$	select $operator_t$		
t+1					

Soar-RL Sequence

Time	Input	Propose	Decide	Apply	Output
t	$state_t$	evaluate $operators_t$	select $operator_t$		initiate action(s)
t+1					

Soar-RL Sequence

Time	Input	Propose	Decide	Apply	Output
t	$state_t$	evaluate $operators_t$	select $operator_t$		initiate action(s)
t+1	$state_{t+1}$ $reward_{t+1}$				

Soar-RL Sequence

Time	Input	Propose	Decide	Apply	Output
t	$state_t$	evaluate $operators_t$	select $operator_t$		initiate action(s)
t+1	$state_{t+1}$ $reward_{t+1}$	evaluate $operators_{t+1}$			

Soar-RL Sequence

Time	Input	Propose	Decide	Apply	Output
t	$state_t$	evaluate $operators_t$	select $operator_t$		initiate action(s)
t+1	$state_{t+1}$ $reward_{t+1}$	evaluate $operators_{t+1}$	select $operator_{t+1}$ update $policy_t$		

Soar-RL Agent Design

To use Soar-RL, an agent must contain two components:

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

Reward Rules

- Upon creation of a new state, Soar creates a **reward-link** identifier

```
<state> ^reward-link <r>
```

- At the beginning of each *decision phase*, Soar collects all properly located numeric constants

```
<state> ^reward-link.reward.value float
```

- 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)
}
```

Controlling Soar-RL

Get/Set a parameter:

```
> rl [-g|--get] <name>
> rl [-s|--set] <name> <value>
```

Soar-RL is **disabled** by default, to enable:

```
> rl --set learning on
```

Debugging Soar-RL

> watch [-R|--rl]

> print [-r|--rl]

`rule*name <number of updates> <current value>`

> preferences

NOTE: selection probabilities

> predict

> select <id>

Left-Right Agent

VisualSoar

$RELEASE/share/soar/Demos/left-right.vsa

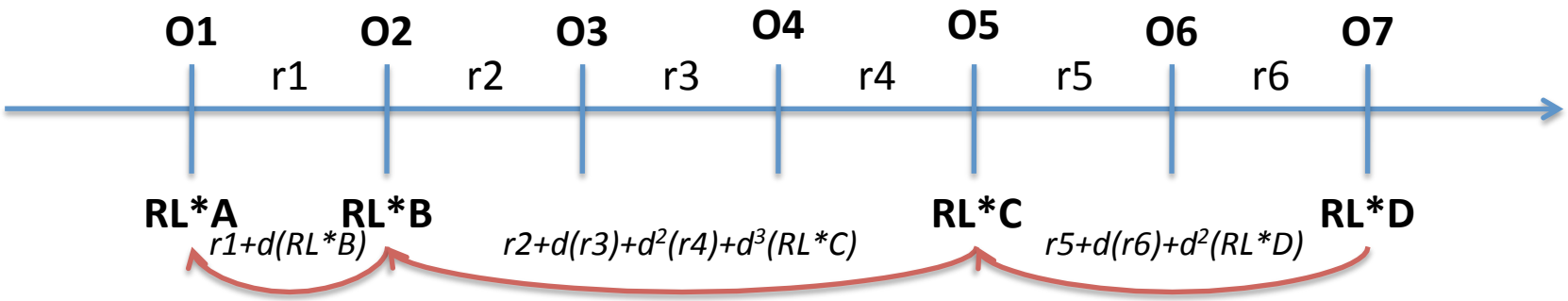
Advanced Topics

- Gaps in rule coverage
- RL in substates

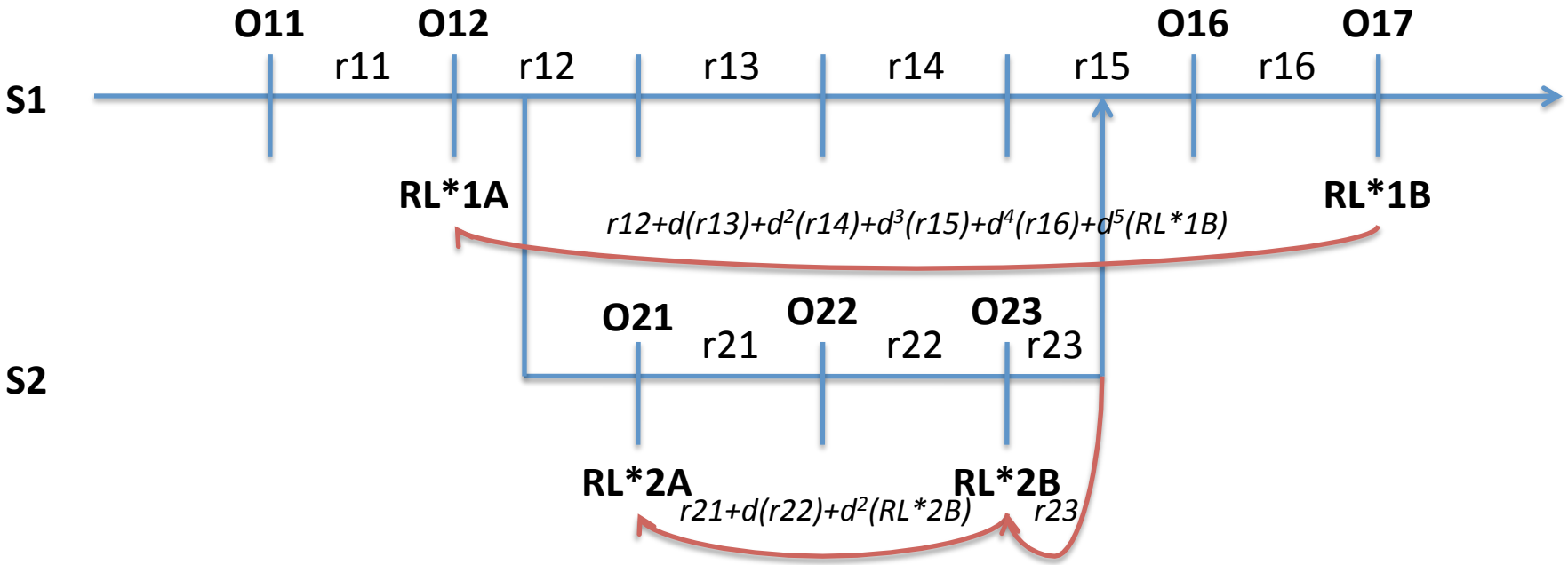
Gaps in Rule Coverage

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, where rewards are discounted with respect to the length of the gap



RL in Substates



- Rewards are collected independently on each state
- Rewards at a superstate are attributed to the last RL-supported, selected operator

Additional Resources

- Soar-RL Documentation
- Soar-RL Demo Agents
- Readings

Soar-RL Documentation

Manual and Tutorial

$SOAR_HOME/share/soar/Documentation

Topics

- Automatically generating Soar-RL rules
- Exploration policies
- Usage: commands, parameters, statistics, etc.

…

Soar-RL Demo Agents

$SOAR_HOME/share/soar/Demos

- Left-Right
- Water Jug RL
  - Discussed in Soar-RL Tutorial
  - Accessible from Soar Java Debugger *Demos* menu
- RL Unit
  - Demonstrates update behavior over gaps/sub-goals

Readings

2004

- Soar-RL: Integrating Reinforcement Learning with Soar
  - Shelley Nason, John E. Laird (ICCM)

2007

- The Importance of Action History in Decision Making and Reinforcement Learning
  - Yongjia Wang, John E. Laird (ICCM)

2008

- A Computational Unification of Cognitive Control, Emotion, and Learning
  - Robert P. Marinier III (Dissertation)

2009

- Learning to Use Episodic Memory
  - Nicholas A. Gorski, John E. Laird (ICCM)
- Learning to Play Mario
  - Shiwali Mohan (Technical Report)
- Hierarchical Reinforcement Learning in the Taxicab Domain
  - Mitchell K. Bloch (Technical Report)

2010

- Instance-Based Online Learning of Deterministic Relational Action Models
  - Joseph Xu, John E. Laird (AAAI)
- Using Imagery to Simplify Perceptual Abstraction in Reinforcement Learning Agents
  - Samuel Wintermute (AAAI)