Programming Reinforcement Learning in Jason

Amelia Bădică[1], Costin Bădică[1], Mirjana Ivanović[2]

[1]University of Craiova, Romania

[2]University of Novi Sad, Serbia

August 26, 2016

16th Workshop on Software Engineering Education
Jahorina, Bosnia and Herzegovina

Talk Outline

- Introduction, Motivation and Related Works
- Temporal Difference Learning
- AgentSpeak(L) and Jason
- Jason Framework for TD Learning
- Results and Discussions
- Conclusions

Motivation

- **Q:** *Experiment with using state-of-the-art AOP languages for agent learning*

Agent Oriented Programming

- Historically, AOP was firstly proposed more than 20 years ago (Shoham, 1990) as:

  A new programming paradigm, one based on cognitive and societal view of computation

- There are many models / implementations of AOP.

Reinforcement Learning

- An RL agent is using the *observed rewards* (known also as *reinforcements*) part of its percepts, to learn an *optimal policy* for acting in an uncertain and dynamic environment (Sutton, 1998).
  - Passive RL: agents act according to a fixed policy and their learning goal is to compute the utility function.
  - Active RL: agents learn an optimal policy that maximizes their utility, while they are acting in their environment.

Benefits of Combining AOP and RL

- From the AOP perspective, RL algorithms can provide a *good benchmark* for experimenting with AOP languages.

- For RL, AOP can be an interesting approach supporting *sound programming principles* for the development of software agent systems.

- Mixing AOP and RL results in new forms of *hybrid reasoning* that benefit by combining different reasoning mechanisms with complementary features into a single cognitive architecture (e.g. *BDI reasoning* and *learning*).

Temporal Difference Learning

- Markovian environment E.
- Stochastic environment.
- The agent strategy is called *policy* $\pi : E \rightarrow A$ where $a = \pi(e)$ is the action carried out by agent in state e of the environment.
- TDL is a *passive RL* method, i.e.:
  - the policy π is given, and
  - the agent's goal is to determine the utilities of states.

Markovian Environment

- The *environment* has a finite set E of states.
- The agent has a finite repertoire A of *available actions*.
- The next environment state *e'* depends only on the current state e and the agent action a ; earlier environment states and agent actions are ignored.
- Some agent states are *terminal* (*final* or *goal*) *states*.

Stochastic Environment

- $p(e' \mid e, a)$ is the *probability* of the environment to transit into state e' given its current state e and agent action a.

- p is a *probability distribution* i.e. $\sum_{e'} p(e' \mid e, a) = 1$ for all $e \in E$ and $a \in A$.

Utility Function

- In each state e of the environment the agent receives a *reward* $R(e) \in \mathrm{IR}$.
- An agent *percept* is a pair $(e, R(e))$.
- The agent utility depends on the rewards received on each state of the environment history $H(e) = [e = e_0, e_1, e_2, \ldots]$:
  - The utility function is *additive* with *discounted rewards*: $U_h([e_0, e_1, e_2, \ldots]) = \sum_{i \geq 0} \gamma^i R(e_i)$, where $\gamma \in (0,1]$ is the *discount factor*.
  - The utility $U^\pi(e)$ for policy π is the *weighted average of the utilities* of each possible environment history $H(e)$ starting in state e, i.e.:

$$U^\pi(e) = \mathbf{E}[U_h(H(e))].$$

Temporal Difference Equation

- When the agent observes a transition $e \rightarrow e'$, (s)he updates $U^{\pi}(e)$ as follows:

  $$U^{\pi}(e) \leftarrow U^{\pi}(e) + \alpha[R(e) + \gamma\, U^{\pi}(e') - U^{\pi}(e)]$$

- α parameter is called *learning rate*. Usually α is monotonically decreasing with the number $n(e)$ of times the environment is in state e.
- TDL does not need to estimate the stochastic model of the environment !

AgentSpeak(L) and Jason

- AgentSpeak(L) is an abstract AOP language, introduced by Rao in 1996.
- Jason is an implementation, as well as an extension of AgentSpeak(L), based on Java.
  - The *agent program* is written in Jason.
  - The *environment* (management of environment state, agent percepts, effect of agent actions) are written in Java.
  - *Agent architecture* can be customized in Java.

Agent Program

- *Belief base* set of Prolog-like facts & rules and it represents the *agent memory*.
- *Plans* define the *agent know-how*. A plan is composed of event, context and body:

  $e : c <- b$

- The belief base is continuously updated by plans during the *agent reasoning cycle*.

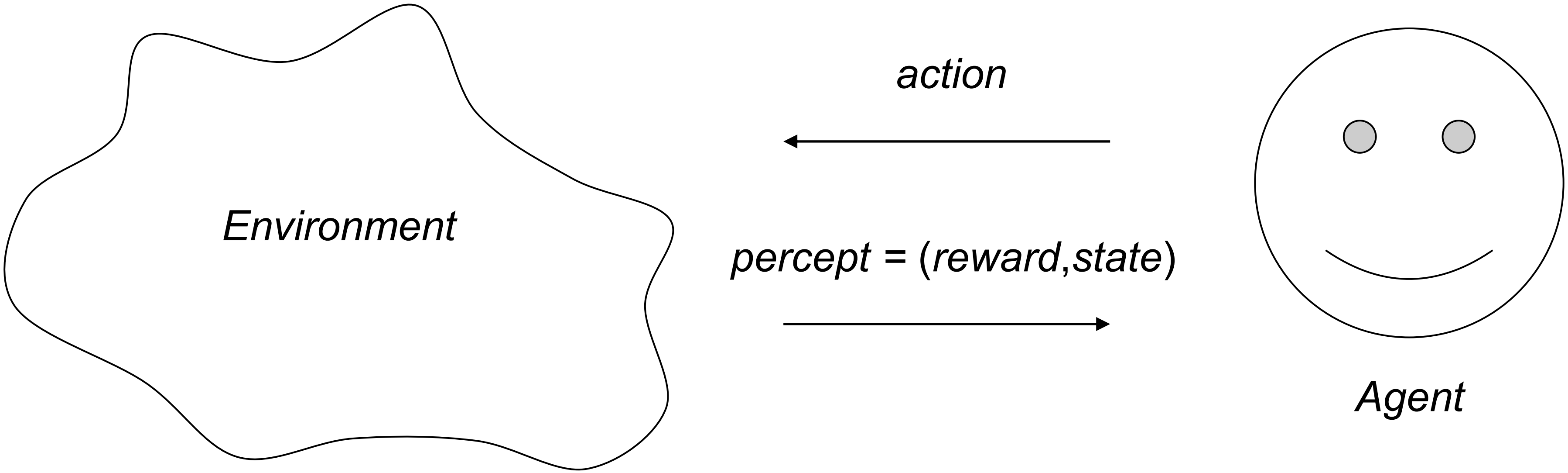
Agent Goals

- The agent is working towards reaching *achievement goals* !*G*.
- *Test goals* ?*G* – used to retrieve information from belief base.
- Using *goals & events* allows to have controllable flexibility in plans.

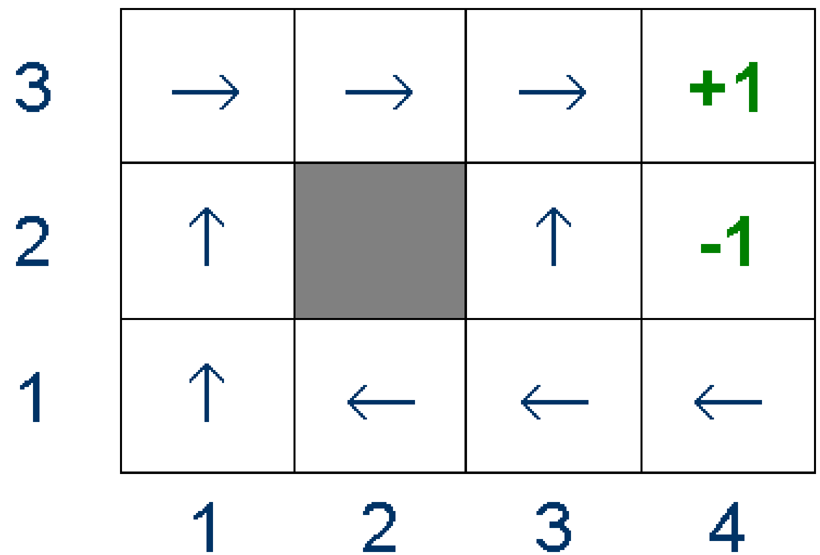
Environment Code

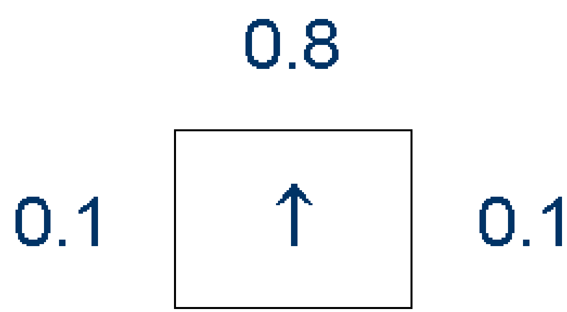
- *Environment* class.
  - *init*() – to initialize the environment.
  - *executeAction*() – to update the environment state after the execution of each agent action (represented as a structured term using *Structure* class
  - *addPercept*() – to add percepts (represented using *Literal* class) to the environment.

Application Model



Sample Environment and Agent Policy





Agent Actions and Percepts

- Agent actions:
  - *up*, *right*, *down*, *left* for the agent movement
  - *null*, for restarting a new trial in a random initial position.
- Agent percepts *pos*(*Row* , *Column* , *R* , *T*) such that:
  - *Row* and *Column* give the agent position on the grid
  - *R* is the reward.
  - *T* is *n* for non-terminal state and *t* for terminal state

Environment Data Structures

- An $m \times n$ grid is represented using a Boolean table *walls* and a real table *rewards* of sizes $0 \ldots m+1 \times 0 \ldots n+1$. They are extra padded with 2 rows and 2 columns representing the grid walls as obstacles.
- Agent actions are represented using *AgAction* enumeration: UP(0), RIGHT(1), DOWN(2), LEFT(3), NULL(4).
- Direction of action is represented using *Direction* enumeration: CHANGE_LEFT(0), KEEP(1), CHANGE_RIGHT(2).
- The offsets of the next agent position from the current agent position is determined based on the agent action and direction, using tables *incRows* and *incColumns*.

Agent Belief Base

- Counter of states: *last_state*(*StateCounter*), initially 0.
- Counter of trials: *last_trial*(*TrialCounter*), initially 0.
- Last executed action: *last_action*(*Action*), initially *null*.
- *terminal_state*(*State*) and *non_terminal_state*(*State*) used in test goals to check if agent's current state is or not a final state.
- Upper bound of number of iterations: represented with *limit*(*UpperBound*) and checked with *below_limit*(*N*).
- Agent's fixed policy: a set of facts *policy*(*State*,*Action*), *State* = *s*(*Row*,*Column*).
- Utility value and number of visits of each state: *utility*(*State*,*UtilityValue*,*Counter*).

Agent Plan Base – !*keep_move*

```
+!keep_move : pos(I,J,R,T) <- // Last perc. pos(I,J,R,T)
   ?last_action(A);  // Determine the last action
   St = s(I,J,T);    // Det. the currently perceived state
   !update(St,R,A);  // Update for reaching state St with act A
   ?last_state(S);
   M = S+1;          // Incr counter for reaching crt state S
   -+last_state(M);
   -+state(M,pos(A,St,R));  // Save last state trans in BB
   !continue_move(M,St).    // continue moving
```

Additional Agent Plans

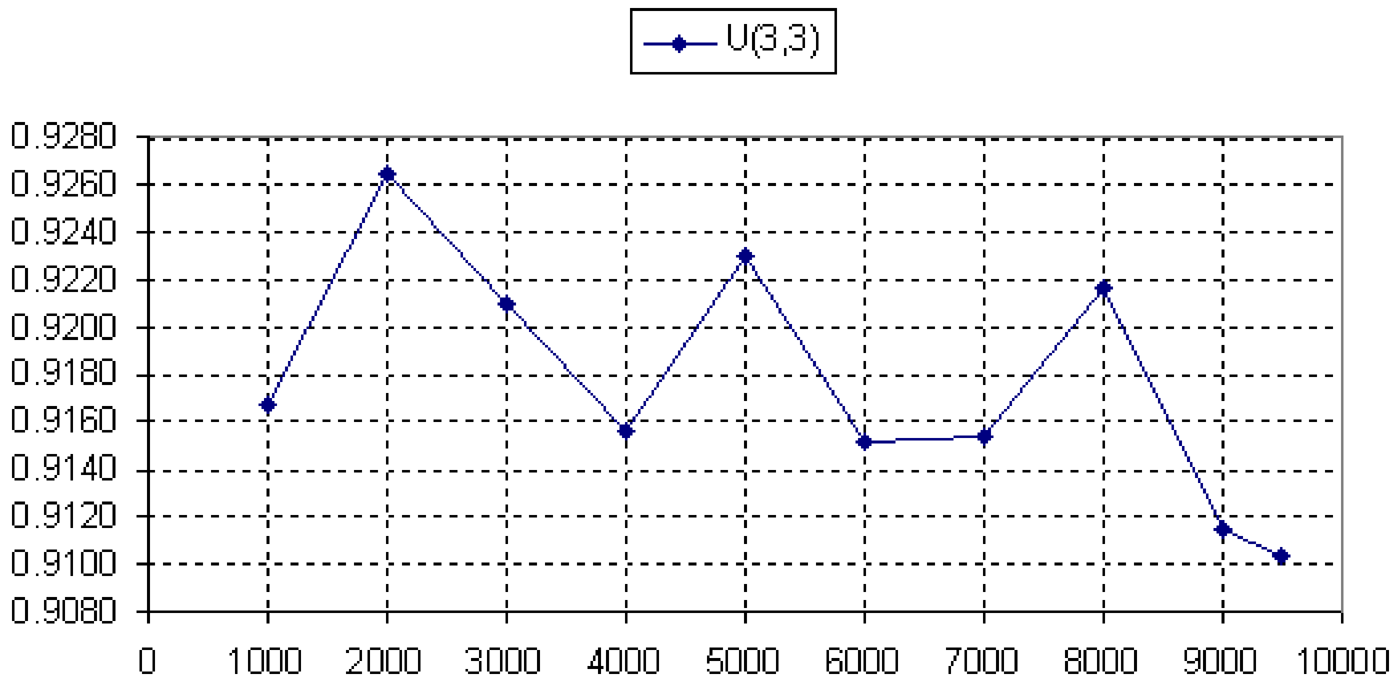
- !*continue_move*(*M*,*St*)
  - Controlled by *below_limit*(*M*) and *terminal_state*(*St*)
  - Take one action by adopting goal !*do_one_move*(*St*)
  - Call !*keep_move*
- !*update*(*St*,R,*A*)
  - Update utility functions

Experiments

- 8 program runs
- 50000 iterations, iteration = state visit
- Number of trials: 9358 (min), 9470.625 (avg), 9528 (max)
- We recorded utility values after each $1000 \times k$ trials, for k = 1, 2, ..., 9, and the end of run

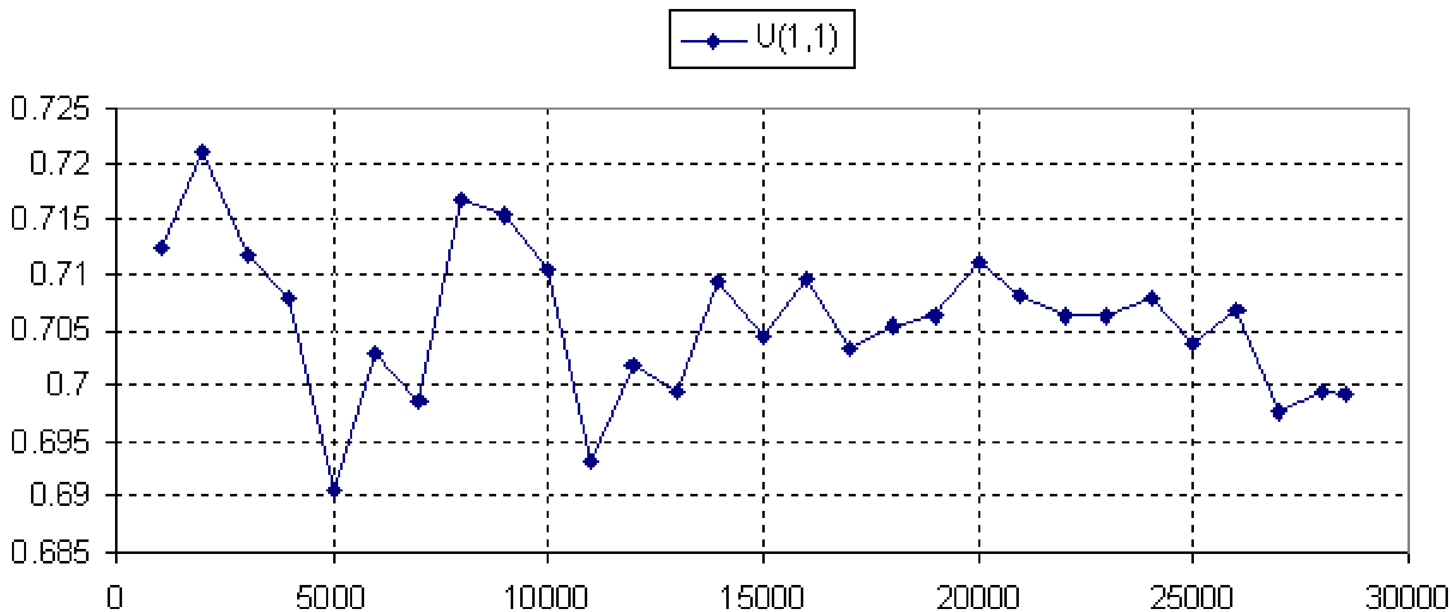
Results

- $U^{\pi}(3;3)$ vs number of trials. The average number of visits is $n(3;3) = 9281.625$



Optimizing Tail Recursion

- We have replaced the tail recursive calls to !*keep_move* with !!*keep_move*, to make the execution of recursive plans more efficient by optimizing tail recursion as iteration.
- 1 run for 150000 of iterations and 28542 trials.



Other Developments

- Experiments with other types of agents:
  - Agents with faulty perception
- A GUI tool for setting up experiments with TDL
- Extending the framework to active RL methods:
  - Q-learning
  - SARSA
- Extending the framework for MAS RL

Conclusion

- Development of a Jason program that implements a Temporal Difference Learning agent.
- Representation of TDL problem using BDI concepts: beliefs, plans and goals.
- Further developments and future works


Thank You