Integrating Reinforcement Learning into a Programming Language

Christopher L. Simpkins
School of Interactive Computing
College of Computing
Georgia Institute of Technology
chris.simpkins@gatech.edu

A Friendly Adaptive Behavior Language

My thesis work combines AI, programming language design, and software engineering. I am integrating reinforcement learning (RL) into a programming language so that the language achieves three primary goals: *accessibility*, *adaptivity*, and *modularity*. My language, AFABL (A Friendly Adaptive Behavior Language), will be an agent programming language designed to be accessible to non-programming experts like behavioral scientists, game designers, and intelligence analysts. If I am successful, my work will enable a discipline of modular large-scale agent software engineering while making advanced agent modeling accessible to authors of agent-based systems who are not programming experts.

There is currently a spectrum of agent-based simulation and programming tools available to social scientists and game designers that runs from the very simple, like NetLogo (Gilbert and Troitsch 2005), to the very powerful, like ABL (A Behavior Language), a state of the art language developed specifically for believable agents (Mateas and Stern 2004). However, the current landscape of agent programming systems suffers from two fundamental and opposing weaknesses. On one hand, simple tools like NetLogo that are accessible to non-programming experts are poorly suited to large-scale agent systems in which the agents are very complex. On the other hand, powerful agent programming languages like ABL are too difficult for non-programming experts to understand and use effectively. With AFABL I intend to close this gap by creating an agent programming language that is accessible to non-programming experts while sacrificing none of the power of advanced languages like ABL.

AFABL will be designed for writing adaptive software agents, that is, agents that learn to adapt to their environment during run-time, not software that is written to be easily changed by modifying the source code and recompiling. I am particularly interested in programming intelligent agents that operate in real environments, and in virtual environments that are designed to simulate real environments. Examples of these kinds of agents include robots, and non-player characters in interactive games and narratives. Unlike traditional programs, agents operate in environments that are often incompletely perceived and constantly changing. This incompleteness of perception and dynamism in the environment creates a strong need for adaptivity. Programming this adaptivity by hand in a language that does not provide built-in support for adaptivity is very cumbersome. As I demonstrated in my 2008 Onward! paper (Simpkins, Bhat, and Isbell 2008), a language with built-in support for adaptivity greatly eases the task of writing adaptive software agents by enabling partial programming, a paradigm in which a programmer or designer specifies the structure of certain parts of a program while leaving other portions unspecified, such that a learning system can learn how to perform them.

If there is one thing that is crucial to the success of my thesis work, it is modular reinforcement learning (MRL). Real-world agents (and agents in interesting artificial worlds) must pursue multiple goals in parallel nearly all of the time. Thus, to make real-world partial programming feasible, we must be able to represent the multiple goals of realistic agents and have a learning system that handles them acceptably well in terms of computation time, optimality, and expressiveness. Typically, multiple-goal RL agents are modeled as collections of RL sub-agents that share an action set. Some arbitration is performed to select the sub-agent action to be performed by the agent. In contrast to hierarchical reinforcement learning, which decomposes an agent's subgoals temporally, we use a formulation of multiple-goal RL which decomposes the agent's subgoals *concurrently*. This concurrent decompositional formulation of multiple-goal RL is better suited to modeling the multiple concurrent goals that must be pursued by realistic agents.

Current Work

While the formulation of MRL described above is intuitive and appealing, providing a practical implementation of it will be challenging for at least two reasons. First, Bhat, et. al., (Bhat, Isbell, and Mateas 2006) have shown that fully general arbitration satisfying a few reasonable constraints is impossible. Second, all current formulations of multiple goal RL (Russell and Zimdars 2003; Sprague and Ballard 2003) assume, either explicitly or tacitly, that sub-agents are authored together and therefore have comparable reward signals. My current work, described below, is focused on solving these two problems.

Bhat's impossibility theorem for ideal arbitration provides a theoretical framework for designing MRL algorithms: the first step in designing a practical MRL algorithm is deciding which constraints of ideal arbitration to relax. Bhat proposes relaxing the non-dictatorship constraint by creating an arbitrator sub-agent whose task is to combine the preferences of the other sub-agents to determine the agent's joint actions. The arbitrator sub-agent gets its own reward signal which represents the desirability of a given action for the agent as a whole, as opposed to the reward signals received by the other sub-agents that represent desirability with respect only to their own subtasks. The idea is that the arbitrator will learn how to combine the sub-agent preferences to achieve a globally optimal policy for the whole agent.

I am taking a fundamentally different approach to MRL. I am interested not in theoretical optimality, but in enabling psychologists, sociologists and game designers to construct agent systems that model personalities in terms familiar to behavioral specialists, not programming specialists. A common theme in the behavioral science literature is that personality is composed of multiple components, with personalities differing in how they weight the different components. (See (Silverman and Bharathy 2005) for a brief computationally-focused overview.) These personality components correspond directly to the components in my agent formulation. Note that I use the term component instead of sub-agent to reflect the fact that an agent is created by combining components, and because I have an eye toward accommodating non-RL components in the future.

Specifying the weights on the components of an agent runs immediately into the problem of reward comparability. If the components of an agent are authored separately, they cannot be expected to use reward values with similar scales. For example, one component might use rewards in the range 1 to 10 while another uses rewards in the range 10 to 100. If these components were combined naively, one would expect the second component to dominate the first in specifying the actions chosen by the agent. The central task of my arbitration algorithm is to learn how to weight these two components automatically, so that the probability that a particular component specifies the action taken by the agent reflects the weights assigned by the agent designer, not the ratios of the reward scales that each component happens to use. This point is crucial to truly modular agent software engineering: components must be able to be written separately without regard to reward comparability.

I expect to have finished developing, analyzing, and testing my MRL algorithm by June 2010 and plan to submit my results to NIPS. In the next section I discuss the remainder of my thesis work which uses the MRL system I just described.

Future Work

While the syntax of AFABL will be different from the examples presented in my previous Onward! paper, the essential idea of adaptive programming remains the same. To finish my dissertation I will flesh out my new language by working with behavioral scientists, simulationists, and game designers. To this end I have been working with a group of industrial and systems engineers led by Dr. Doug Bodner at Georgia Tech's Tennenbaum Institute who are creating tools for agent-based organizational simulations (Bodner and Rouse 2009), and I plan to work with Georgia Tech computing professor Mark Riedl in his interactive narrative projects (Riedl and Stern 2006; Riedl et al. 2008). Both of these collaborations will provide case studies in which I can compare and contrast agent systems written in ABL with agent systems written in AFABL. These collaborations will provide two benefits. First, working with agent modelers will help me design the language to meet the needs of the kinds of users I am targeting. Second, their agent systems will provide a test bed to validate the design of my language. I hope to show by detailed side-by-side comparison that AFABL significantly reduces the semantic gap between the programming concepts an agent designer must understand to effectively write agent programs, and the behavioral concepts a non-programmer behavior specialist normally uses to describe intelligent agents such as humans and organizations.

References

Bhat, S.; Isbell, C.; and Mateas, M. 2006. On the difficulty of modular reinforcement learning for real-world partial programming. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06).*

Bodner, D. A., and Rouse, W. B. 2009. *Handbook of Systems Engineering and Management*. Wiley. chapter Organizational Simulation.

Gilbert, N., and Troitsch, K. G. 2005. *Simulation for the Social Scientist*. Berkshire, England: Open University Press, McGraw-Hill Education.

Mateas, M., and Stern, A. 2004. *Life-like Characters. Tools, Affective Functions and Applications*. Springer. chapter A Behavior Language: Joint Action and Behavioral Idioms.

Riedl, M. O., and Stern, A. 2006. Believable agents and intelligent scenario direction for social and cultural leadership training. In *Proceedings of the 15th Conference on Behavior Representation in Modeling and Simulation*.

Riedl, M. O.; Stern, A.; Dini, D.; and Alderman, J. 2008. Dynamic experience management in virtual worlds for entertainment, education, and training. In *International Transactions on Systems Science and Applications, Special Issue on Agent Based Systems for Human Learning*, volume 4(2).

Russell, S., and Zimdars, A. L. 2003. Q-decomposition for reinforcement learning agents. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003).*

Silverman, B. G., and Bharathy, G. K. 2005. Modeling the personality and cognition of leaders. Technical report, University of Pennsylvania, Philadelphia, PA.

Simpkins, C.; Bhat, S.; and Isbell, C. 2008. Towards adaptive programming: Integrating reinforcement learning into a programming language. In *OOPSLA '08: ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, Onward! Track.*

Sprague, N., and Ballard, D. 2003. Multiple-goal reinforcement learning with modular sarsa.