Towards Adaptive Programming: Integrating Reinforcement Learning into a Programming Language

Chris Simpkins[1],

Sooraj Bhat[1],

Charles Isbell[1],

Michael Mateas[2]

[1]Georgia Tech, [2]UC Santa Cruz

The Road to Adaptive Software:

Reinforcement Learning + Programming Language = Partial Programming

- Illustrate problems with concrete example
- Propose and define solution: adaptive programming
- Illustrate adaptive programming via partial programming paradigm

The Problem

- In the context of interactive games, narratives, and training simulations, adaptive agent programming demands more than current programming paradigms can deliver
  - Agents must be able to adapt to different environments without reprogramming, which is difficult or impossible with the current state of the art in software engineering
- I'll illustrate the problem with a simple concrete example in a cutting-edge agent language, ABL

Example: Furry Creature Agent

(0, 4)	\ / ^			(4, 4)
				
		^–^ oo >–<		
(0, 0)				(4, 0)

Actions: MoveNorth, MoveSouth, MoveEast, MoveWest

A Furry Creature in ABL

Overarching goal LiveLongProsper achieved by pursuing two subgoals in parallel

```
behaving_entity FurryCreature
{
  parallel behavior LiveLongProsper() {
    subgoal FindFood();
    subgoal AvoidPredator();
  }

  sequential behavior FindFood() {
    with (ignore_failure) subgoal MoveNorthForFood();
    with (ignore_failure) subgoal MoveSouthForFood();
    with (ignore_failure) subgoal MoveEastForFood();
    with (ignore_failure) subgoal MoveWestForFood();
  }

  sequential behavior MoveNorthForFood() {
    precondition {
      (FoodWME x::foodX y::foodY)
      (SelfWME x::myX y::myY)
      ((foodY - myY) > 0) // The food is north of me
    }

    // Code for moving agent to the north elided
  }
//...
}
```

A Furry Creature in ABL

```
behaving_entity FurryCreature
{
  parallel behavior LiveLongProsper() {
    subgoal FindFood();
    subgoal AvoidPredator();
  }

  sequential behavior FindFood() {
    with (ignore_failure) subgoal MoveNorthForFood();
    with (ignore_failure) subgoal MoveSouthForFood();
    with (ignore_failure) subgoal MoveEastForFood();
    with (ignore_failure) subgoal MoveWestForFood();
  }

  sequential behavior MoveNorthForFood() {
    precondition {
      (FoodWME x::foodX y::foodY)
      (SelfWME x::myX y::myY)
      ((foodY - myY) > 0) // The food is north of me
    }

    // Code for moving agent to the north elided
  }
//...
}
```

Overarching goal LiveLongProsper achieved by pursuing two subgoals in parallel

FindFood is achieved by some sequence of 4 behaviors. Reactive planner selects behaviors based on preconditions ...

A Furry Creature in ABL

```
behaving_entity FurryCreature
{
  parallel behavior LiveLongProsper() {
    subgoal FindFood();
    subgoal AvoidPredator();
  }

  sequential behavior FindFood() {
    with (ignore_failure) subgoal MoveNorthForFood();
    with (ignore_failure) subgoal MoveSouthForFood();
    with (ignore_failure) subgoal MoveEastForFood();
    with (ignore_failure) subgoal MoveWestForFood();
  }

  sequential behavior MoveNorthForFood() {
    precondition {
      (FoodWME x::foodX y::foodY)
      (SelfWME x::myX y::myY)
      ((foodY - myY) > 0) // The food is north of me
    }

    // Code for moving agent to the north elided
  }
//...
}
```

Overarching goal LiveLongProsper achieved by pursuing two subgoals in parallel

FindFood is achieved by some sequence of 4 behaviors. Reactive planner selects behaviors based on preconditions ...

Preconditions coded using pattern matching and pattern guards. Working Memory Element (WME) is the sensory mechanism for ABL agents

A Furry Creature in ABL

```
behaving_entity FurryCreature
{
  parallel behavior LiveLongProsper() {
    subgoal FindFood();
    subgoal AvoidPredator();
  }

  sequential behavior FindFood() {
    with (ignore_failure) subgoal MoveNorthForFood();
    with (ignore_failure) subgoal MoveSouthForFood();
    with (ignore_failure) subgoal MoveEastForFood();
    with (ignore_failure) subgoal MoveWestForFood();
  }

  sequential behavior MoveNorthForFood() {
    precondition {
      (FoodWME x::foodX y::foodY)
      (SelfWME x::myX y::myY)
      ((foodY - myY) > 0) // The food is north of me
    }

    // Code for moving agent to the north elided
  }
//...
}
```

Overarching goal LiveLongProsper achieved by pursuing two subgoals in parallel

FindFood is achieved by some sequence of 4 behaviors. Reactive planner selects behaviors based on preconditions ...

Preconditions coded using pattern matching and pattern guards. Working Memory Element (WME) is the sensory mechanism for ABL agents

Problems:

- Assumes world state known a priori
- Programmer resolves subgoal conflicts - haven't shown conflict resolution code in this example
- Actions and selection logic coupled - *what* entangled with *how*. 8 separate behaviors needed for simple FurryCreature in ABL (2*4). Add a 3rd subgoal and you need 12 (3*4)

Solution: Adaptive Programming

- Norvig and Cohn's definition:
  - Adaptive software uses available information about changes in its environment to improve its behavior (Norvig and Cohn 1998)
- We seek software that adapts at run-time, not software designed to be modified and recompiled for different environments

Adaptive Software via Partial Programming

- What we need: Partial Programming
  - Code partial solutions, run-time system learns remainder of solution *at run-time*
    - Separates the *what* of agent behavior from the *how* - I'll show you how with an A^{2}BL Furry Creature
- How we'll get it: integrate reinforcement learning into a programming language

Reinforcement Learning

- States - agent can be in one of a finite number of states
- Actions - agent has a set of actions available in each state that affects the state of the world, i.e., moves the agent to a different state
- Rewards - each state gives the agent feedback in the form of a scalar reward signal
- RL algorithm learns control policy - given a state, what action leads to best long-term cumulative reward, i.e., more rewarding actions are *reinforced*

Example: Furry Creature Agent

s_{20} r = 0	\ / ^ r = -100	s_{22} r = 0	s_{23} r = 0	s_{24} r = 0
s_{15} r = 0	s_{16} r = 0	s_{17} r = 0	s_{18} r = 0	 r = 100
s_{10} r = 0	s_{11} r = 0	^-^ oo >-<	s_{13} r = 0	s_{14} r = 0
s_5 r = 0	s_6 r = 0	s_7 r = 0	s_8 r = 0	s_9 r = 0
s_0 r = 0	s_1 r = 0	s_2 r = 0	s_3 r = 0	s_4 r = 0

Actions: MoveNorth, MoveSouth, MoveEast, MoveWest

A Furry Creature in A²BL

```
behaving_entity FurryCreature
{
  adaptive collection behavior LiveLongProsper() {
    subgoal FindFood();
    subgoal AvoidPredator();
  }

  // subgoal 1
  adaptive sequential behavior FindFood() {
    reward {
      100 if { (FoodWME) }
    }
    state {
      (FoodWME x::foodX y::foodY)
      (SelfWME x::myX y::myY)
      return (myX,myY,foodX,foodY);
    }
    subgoal MoveNorth();
    subgoal MoveSouth();
    subgoal MoveEast();
    subgoal MoveWest();
  }
// ...
}
```

`adaptive collection` behavior - run-time system pursues both subgoals in parallel *and resolves conflicts between them*

A Furry Creature in A²BL

```
behaving_entity FurryCreature
{
  adaptive collection behavior LiveLongProsper() {
    subgoal FindFood();
    subgoal AvoidPredator();
  }

  // subgoal 1
  adaptive sequential behavior FindFood() {
    reward {
      100 if { (FoodWME) }
    }
    state {
      (FoodWME x::foodX y::foodY)
      (SelfWME x::myX y::myY)
      return (myX,myY,foodX,foodY);
    }
    subgoal MoveNorth();
    subgoal MoveSouth();
    subgoal MoveEast();
    subgoal MoveWest();
  }
// ...
}
```

`adaptive collection` behavior - run-time system pursues both subgoals in parallel *and resolves conflicts between them*

`adaptive sequential` behavior - run-time system learns how to sequence the subgoals

A Furry Creature in A²BL

```
behaving_entity FurryCreature
{
  adaptive collection behavior LiveLongProsper() {
    subgoal FindFood();
    subgoal AvoidPredator();
  }

  // subgoal 1
  adaptive sequential behavior FindFood() {
    reward {
      100 if { (FoodWME) }
    }
    state {
      (FoodWME x::foodX y::foodY)
      (SelfWME x::myX y::myY)
      return (myX,myY,foodX,foodY);
    }
    subgoal MoveNorth();
    subgoal MoveSouth();
    subgoal MoveEast();
    subgoal MoveWest();
  }
// ...
}
```

`adaptive collection` behavior - run-time system pursues both subgoals in parallel *and resolves conflicts between them*

`adaptive sequential` behavior - run-time system learns how to sequence the subgoals

`reward` constructs specify rewards the agent receives in particular states

A Furry Creature in A²BL

```
behaving_entity FurryCreature
{
  adaptive collection behavior LiveLongProsper() {
    subgoal FindFood();
    subgoal AvoidPredator();
  }

  // subgoal 1
  adaptive sequential behavior FindFood() {
    reward {
      100 if { (FoodWME) }
    }
    state {
      (FoodWME x::foodX y::foodY)
      (SelfWME x::myX y::myY)
      return (myX,myY,foodX,foodY);
    }
    subgoal MoveNorth();
    subgoal MoveSouth();
    subgoal MoveEast();
    subgoal MoveWest();
  }
// ...
}
```

`adaptive collection` behavior - run-time system pursues both subgoals in parallel *and resolves conflicts between them*

`adaptive sequential` behavior - run-time system learns how to sequence the subgoals

`reward` constructs specify rewards the agent receives in particular states

`state` constructs specify the part of the world state that this behavior cares about

A Furry Creature in A²BL

```
behaving_entity FurryCreature
{
  adaptive collection behavior LiveLongProsper() {
    subgoal FindFood();
    subgoal AvoidPredator();
  }

  // subgoal 1
  adaptive sequential behavior FindFood() {
    reward {
      100 if { (FoodWME) }
    }
    state {
      (FoodWME x::foodX y::foodY)
      (SelfWME x::myX y::myY)
      return (myX,myY,foodX,foodY);
    }
    subgoal MoveNorth();
    subgoal MoveSouth();
    subgoal MoveEast();
    subgoal MoveWest();
  }
// ...
}
```

`adaptive collection` behavior - run-time system pursues both subgoals in parallel *and resolves conflicts between them*

`adaptive sequential` behavior - run-time system learns how to sequence the subgoals

`reward` constructs specify rewards the agent receives in particular states

`state` constructs specify the part of the world state that this behavior cares about

Run-time RL system automatically learns how to sequence subgoal actions based on the states and rewards defined for the enclosing behavior

Advantages of A²BL's Built-in Adaptivity

- World state is learned by the agent, but specified separately, possibly by a different programmer - a way for subject matter expert to inject domain knowledge
- Conflicts between subgoals automatically resolved by A²BL's modular RL
- Decouples *what* from *how* - add a new subgoal and you only have to code it *once*

Norvig and Cohn's Model of Adaptive Programming

Traditional	**Adaptive**	**ABL**	**A²BL**
Function/Class	Agent/Module	Yes	Yes
Input/Output	Perception/Action	Yes	Yes
Logic-based	Probability-based	No	**Yes**
Goal-based	Utility-based	No	**Yes**
Sequential, single-	Parallel, multi-	Yes	Yes
Hand-programmed	Trained (Learning)	No	**Yes**
Fidelity to designer	Perform well in environment	Sorta	Yes
Pass test suite	Scientific method	Sorta	Yes

Future Directions

- Usability - visual programming, high- vs. low-level abstractions
- Structuring mechanisms for large systems
- Validation
- Run-time efficiency
- What is the meaning of "debugging" in the setting of a learning agent?

Future Directions - OOP

- Agent's are kinda like objects, but different
- What is a subtype in the agent world?
- What does "is-a" mean?
  - Inherit learning algorithm, learned model?
  - Contracts in stochastic realm?
- What does "has-a" mean?
- Intellectually fun - we get to explore the very notion of agency

Thanks!

simpkins@cc.gatech.edu

I'll be here until tomorrow afternoon. Come find me if you'd like to talk.