UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

EFFECTS OF STATE AND ACTION ABSTRACTION ON DEVELOPMENT OF CONTROLLERS FOR CONCURRENT, INTERFERING, NON-EPISODIC TASKS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

BRENT E. ESKRIDGE
Norman, Oklahoma
2009

EFFECTS OF STATE AND ACTION ABSTRACTION ON DEVELOPMENT OF CONTROLLERS FOR CONCURRENT, INTERFERING, NON-EPISODIC TASKS

A DISSERTATION APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

Dr. Dean F. Hougen, Chair

Dr. John K. Antonio

Dr. Sesh Commuri

Dr. Andrew H. Fagg

Dr. Amy McGovern

For Maya, Connor, and Noah

Acknowledgements

Words do not adequately describe how grateful I am to my family for their support and understanding throughout this entire journey. My wife, Maya, has had to endure all those nights when I came home from work only to retreat to the office to perform research. Without her strength, I would have never survived. Although my sons Connor and Noah are too young to understand what was happening, their simple smiles and laughs reminded me why I was doing this and gave me the strength to continue. I can't wait to spend this summer actually playing with you and not working every minute of the day.

I am also thankful for all that parents have done to help. From the kind words of support to the more tangible expressions of support, you have been there every step of the way and never doubted me. Thanks for all your help to me and my family. Thanks also go to my sister, Teri, and her family, Mark, Anderson, and Jackson. I appreciate all you have done over the years from the small to the big.

I am immensely grateful to my advisor Dean Hougen. His support and confidence in me has made me into a better computer scientist. Thank you for your willingness to give me freedom to pursue my own interests and the guidance which helped me turn dead-ends into opportunities. I also appreciate your flexibility in working with my hectic work schedule and "distance" learning. Beyond research, I thank you for your friendship. Without the times of laughing about reviewers or students, I'm sure I would have lost all of my remaining sanity.

I also wish to thank Amy McGovern for her insights and assistance both inside the classroom and beyond. I have valued all our discussions. I have no doubt that Nathaniel is in good hands. I'm also grateful to Andy Fagg for his help and guidance. Your attention to detail has instilled a healthy fear in me regarding the use of statistics in my results. Thanks

also go to the remaining members of my committee, Sesh Commuri and John Antonio. I appreciate your willingness to contribute to my research even though this area isn't your specialty.

I am also grateful to the members of the Robotics, Evolution, Adaptation and Learning Laboratory for their support and insightful comments: Mark Woehrer, Pedro Diaz-Gomez, Jason Black, and Gerardo Gonzalez. While I may not have been in the lab as much as I would have liked, I appreciate all the opportunities I had to work with you. Since I will be remaining close by, I sincerely hope we can continue to collaborate.

None of my work would have been possible were it not for the support and understanding of my colleagues at Southern Nazarene University. The faculty and administration supported my every effort, be it coursework or research. I am grateful for all you have done. Special thanks go to Jim Tabers and Gwen Rodgers who helped pick up my slack and Dwight Neuenschwander who always offered a motivational word. Thanks also go to all my students for your understanding of my hectic schedule and forgetting to grade your homework (although in some cases that wasn't a bad thing).

Some of the computing for this project was performed at the OU Supercomputing Center for Education & Research (OSCER) at the University of Oklahoma.

Table of Contents

List of Tables

List of Figures

Abstract

The development of controllers for autonomous intelligent agents given a simple task is relatively straightforward and basic techniques can be used to develop such controllers. However, as agents are given more than one task, using basic techniques for developing effective controllers quickly becomes impractical. State and action abstraction are frequently used to counter this explosion of complexity and to make the development of effective controllers for complex problems practical. Unfortunately, most of the work in the literature has focused on complex tasks comprised of sequences of simpler tasks and the more complex tasks comprised of many concurrent, interfering, and non-episodic (CINE) tasks have received little attention. As a result, this dissertation seeks to address this deficiency by providing the first known empirical investigation into the effects of each of these types of abstraction on CINE tasks. The results of this investigation demonstrate that for the single-agent and multi-agent problem domains used, abstraction of the controller's actions provides more benefits in the development and performance of effective controllers than abstraction of the agent's state.

Since there is a lack of work focusing on complex CINE tasks, advances in the implementation and development of controllers capable of addressing such tasks were required. First, we demonstrate that the adaptive fuzzy behavior hierarchy control architecture used in this dissertation has issues when scaled to hierarchies of more than two levels. To address these issues, we introduce a modification to the architecture's implementation that significantly improves the performance of controllers using the same behavior hierarchy. Second, we demonstrate that one of the few known reinforcement learning approaches specifically designed to handle complex CINE tasks is unable to converge to an effective policy for the tasks used here. As a result, we introduce a new reinforcement learning ap-

proach that leverages the hierarchical implementation of the controller which is capable of providing statistically significantly better performance in significantly fewer learning experiences. Next, we demonstrate that controllers using adaptive fuzzy behavior hierarchies are able to reuse, without modification, controllers developed for simple tasks in hierarchical controllers developed for a more complex task. Lastly, we demonstrate that since adaptive fuzzy behavior hierarchies effectively use action abstraction, the agent's state can be significantly abstracted in the higher levels of the controller using adaptive priorities which reflect the applicability of lower level behaviors to the agent's current state.

CHAPTER 1

Introduction

The development of controllers for intelligent agents given a simple task is relatively straightforward and even basic techniques can be used to develop such controllers. However, as agents are given more than one task, using basic techniques for developing effective controllers quickly becomes impractical. Since each task may require distinct state information local to that task only, the resulting state space for the agent overall is simply too large to effectively cover. Furthermore, since each task places different demands on the agent, an effective controller must find the correct balance to achieve the overall task. While each of these difficulties present significant problems individually, their combination can make the development of agent controllers for these complex tasks impractical.

The potential avenues of investigation for such a problem are vast and the literature on the subject covers the spectrum of algorithms and perspectives. This dissertation's focus is on complex, or *composite*, tasks that are the result of a combination of, in general, **C**oncurrent, **I**nterfering, and **N**on-**E**pisodic (CINE) simple, or primitive, tasks. An example of a such a primitive task is that of *collision avoidance*. This area has received comparatively little attention and is, for reasons discussed below, potentially one of the more difficult areas of focus. In contrast, many approaches focus on complex tasks that are composed of a series of sequential, simple tasks.

Each of the attributes of CINE primitive tasks presents unique challenges that make many of the more common approaches inapplicable. If the primitive tasks were not active concurrently, then a controller could simply focus on the currently active task. However, since the tasks are active concurrently, a controller must take actions that balance the needs

of each of the active tasks. An example of concurrent primitive tasks would be giving a robot the composite task of navigating to a goal location while avoiding moving obstacles in its path where *goal seek* and *collision avoidance* are primitive tasks. In contrast, an example of sequential primitive tasks would be to navigate to goal location 1 and then navigate to goal location 2. If the actions taken for primitive tasks did not interfere with one another, then the controller could simply take the action that produces the best result for each primitive task. However, since the actions that are best for individual primitive tasks do, in general, interfere with one another, the controller must take the action that best accomplishes the composite task, even if it is at the expense of an individual primitive task. Continuing with the previous example, the goal seek task might be best addressed by steering straight ahead, but since an obstacle is in the way, the collision avoidance task would be best addressed by steering to the left. Lastly, if the primitive tasks were episodic and had termination criteria, a controller could use temporal abstraction to combine a sequence of actions into a single meta-action that accomplishes a task. However, since the primitive tasks are, in general, non-episodic, temporal abstraction cannot be applied.[1] In the previous example, the goal seek task is episodic since the task is completed once the robot reaches the goal. However, the collision avoidance task is non-episodic since it is always active and, regardless of the other tasks the robot may have, it is always desirable to avoid collisions.

These challenges, in addition to the complexity of the state and action spaces, must be addressed for the development of effective controllers for combinations of CINE tasks to be practical. Due to the variety of challenges, it is possible that a number of techniques must be combined to find an acceptable solution. While temporal abstraction is not useful for CINE tasks, there are other abstraction approaches that can be used. State abstraction is a common technique used to simplify the state space used by the agent controller. Abstraction of the action space can also be performed in a manner similar to that found in temporal

[1] However, Huber describes a system where it can be argued that temporal abstraction can be applied to a non-episodic behavior [34].

abstraction. However, at some point, the controller must have access to unabstracted states and actions in order to take the most appropriate action. For example, a controller cannot simply use the magnitude of the direction to the goal location and be expected to effectively navigate towards the goal.

One approach that promotes the use of state and action abstraction while still allowing access to the unabstracted states and actions is the use of a hierarchical controller. A hierarchical controller leverages the hierarchical nature of the composite task by using smaller controllers responsible for each primitive task in the lowest level of the hierarchical controller and meta-controllers in the higher levels of the hierarchical controller to coordinate the lower-level controllers. Since low-level controllers are only responsible for a single primitive task, they do not need access to the full state space of the composite task, thus avoiding the combinatorial complexity of the composite task's state space. Furthermore, high-level meta-controllers are able to use state and action abstraction to simplify the state space since they merely coordinate the lower-level controllers that produce control actions instead of producing control actions themselves.

While various combinations of these approaches can be found in the literature, we have been unable to find a systematic, empirical evaluation and comparison of the effects of state and action abstraction on the practicality of the ability to develop controllers for these tasks and on the resulting performance. The work presented in this dissertation attempts to address this deficiency by asking the following experimental questions:

> Given a composite task in which the subtasks are, in general, concurrent, interfering, and non-episodic (CINE),
>
> 1. what are the effects of abstracting state information on the performance and development rate of controllers, and
> 2. what are the effects of abstracting actions on the performance and development rate of controllers?

The evidence gained by attempting to answer this question provides a number of significant

benefits. First, it can provide valuable insight for controller developers into how much abstraction is appropriate for a particular situation. For example, in one situation, a minor loss in performance may be acceptable if the effort in developing an effective controller is significantly reduced, while, in another situation, such a loss in performance could be unacceptable no matter what the additional benefits were. Another benefit is that these results can inform controller developers as to which approach, state or action abstraction, offers the most potential in a particular situation. Lastly, these results can provide deeper understanding into whether the effects of state and action abstraction are a function of the number of primitive tasks.

1.1 Research Motivation and Summary

The research presented in this dissertation was motivated by a need to make practical the development of autonomous agent controllers used in complex, composite tasks. To narrow the focus, we concentrated on composite tasks comprised of multiple CINE primitive tasks. This research attempted to gain insight into the effects of state and action abstraction on the performance and development of such controllers. To minimize the number of possible sources that can affect performance, the focus was further narrowed to tasks that only require reactive behaviors and do not need planning. If planning were required, it is quite likely that the planner would affect performance and could potentially do so differently based on the state and action abstractions used. Although other work may not choose to be as restrictive in their definition of a purely reactive architecture, we use a definition that specifically excludes planning and memory to prevent conflating the many variables that they introduce [56]. Furthermore, we do not provide an *a priori* prioritization of the primitive tasks as this would simplify the control process and we are interested in problem domains that are as complex as possible.

While changes in performance can be easily measured, the relative rates and difficulties in developing such controllers is not so easily quantifiable. As a result, we used two differ-

ent approaches for automatically developing controllers and use the computational effort required to produce effective controllers as a basis of comparison. While the parameters, environments, and reward structures used could have been tailored for each approach to bias development of the desired behaviors, we chose not to implement any such bias in order to isolate the effects of state and action abstraction.

Although there are a number of potential approaches to gaining insight into these effects, we chose to focus on a single approach and used it to perform an in-depth analysis. In this work, we evaluated the effects of state and action abstraction on an adaptive fuzzy behavior control architecture proposed by Tunstel [93]. It was designed to handle CINE tasks such as the ones used in this dissertation. Furthermore, it allowed us to evaluate the effects of state and action abstraction on controllers that reused previously developed sub-controllers since such an ability is particularly desirable in developing controllers for CINE tasks. Note that since we start from a set of primitive tasks which we combine into a composite task, we do not need to find a functional decomposition of the composite task as we would need to do if we instead started with the composite task itself [6].

A fundamental component of Tunstel's architecture is the use of fuzzy logic for control [20]. Fuzzy control gives the controller the ability to view the state and action space of the agent as discretized while retaining the continuous nature of the values. As a result, fuzzy control offers the improved and smooth control actions found in continuous controllers with the comparatively less complicated implementation found in discrete controllers. In light of these benefits and Tunstel's extensive use of fuzzy control, all the controllers used in this work to evaluate the effects of state and action abstraction use fuzzy control in their implementation. However, controllers developed for comparison purposes, which use approaches other than Tunstel's, do not use fuzzy logic.

1.2 Contributions

The work presented in this dissertation contains a number of contributions to the fields machine learning and agent control. As in any work, some of the contributions are minor and provide a foundation and motivation for the more significant contributions. Without these minor contributions, the need for and direction of the resulting major contributions might never be realized. Therefore, we detail both types of contributions with the hope that they may provide the launching point for further contributions.

Some of the major contributions of this work are:

- To address the two-level hierarchy limitation in adaptive fuzzy behavior hierarchies (discussed below), we developed an extension that allows for the use of hierarchies with an arbitrary number of levels. Furthermore, we demonstrate that the resulting hierarchies have significantly higher performance and are developed significantly faster than hierarchies without this extension.

- To push state abstraction to its limits, we developed an approach in which the entire state local to a primitive task is abstracted into a single variable, referred to as a *adaptive, dynamic priority*. The value reflects the current priority of the primitive task given information local only to the primitive task. This adaptive, dynamic priority is then used by the higher-level meta-controllers to control the lower-level sub-controllers [23, 24].

- As is discussed in Section 2.3, there is a lack of reinforcement learning approaches that are tailored to problem domains using CINE tasks. Furthermore, the few approaches that are applicable to CINE tasks are not able to produce effective polices for the problem domains used here. In response, we developed a reinforcement learning approach, called *composite* reinforcement learning, which leverages the adaptive behavior hierarchy architecture to learn polices for CINE tasks. As a result, poli-

cies learned by composite reinforcement learning have higher performance and are learned faster than other approaches.

- Due to the hierarchical nature of composite tasks, the ability to reuse existing sub-controllers in a new composite task can provide significant advantages in the development of effective controllers. In the results presented in this dissertation, we show that existing primitive task sub-controllers can be reused without modification by an adaptive fuzzy behavior hierarchy for use in a new composite task.

- As previously discussed, we have been unable to find an empirical investigation into the effects of state and action abstraction in the development of controllers for complex CINE tasks. As a result, this dissertation presents the only results of just such an investigation of which we are aware.

- The most significant aspect of this dissertation's results is that the ability to abstract the action space provides more benefits in practical development of effective agent controllers than state abstraction in the problem domains and CINE tasks evaluated. As previously stated, this is the only known investigation which directly compares the two methods of abstraction.

Some of the minor contributions of this work are:

- One approach for learning policies for CINE tasks is modular reinforcement learning [36, 40]. We demonstrate that modular reinforcement learning is incapable of learning and converging to effective policies in the navigation problem domains used in this research. Previous work in the literature attempted to determine if convergence was guaranteed and was unable to provide an answer [84].

- The literature details a number of examples where genetic algorithms and genetic programming have been used to evolve fuzzy rulesets for agent control. However, a new approach, grammatical evolution [58, 79], which uses the representation of

genetic algorithms and the expressive power of genetic programming has not been previously shown capable of evolving such rulesets. We demonstrate that grammatical evolution can indeed evolve effective fuzzy rulesets.

- While adaptive fuzzy behavior hierarchies have been shown to be an effective control architecture, we identified a deficiency in the fuzzy inferencing equations, which limits its effectiveness to hierarchies of only two levels. In a set of experiments, we demonstrate that this deficiency can be detrimental to the performance and development of effective control hierarchies.

1.3 Organization of Thesis

This chapter has introduced the motivation for and a brief summary of the work presented in this dissertation. The rest of the dissertation is organized as follows:

Chapter 2 presents work related to the motivations of this dissertation. An emphasis is placed on foundational work relevant to this dissertation and how it differs from existing approaches. Furthermore, numerous examples of methods which address the complexity inherent to the automatic development of agent controllers are provided.

Chapter 3 introduces the agent controller architecture used in this dissertation. Since adaptive fuzzy behavior hierarchies use fuzzy inferencing techniques to implement a hierarchy of behaviors, a short introduction to fuzzy logic and its usefulness in control is presented. A limitation of the architecture in its application to complex hierarchies is described along with our extension which removes this limitation. Finally, details regarding our particular implementation of the architecture are provided.

Chapter 4 discusses the problem domains used in this dissertation to evaluate the effect of state and action abstraction in the practicality of learning autonomous agent controllers. In the first set of problem domains, a single-agent is tasked with navigating

through an environment to a goal location. In the second set of problem domains, a group of agents is tasked with navigating through an environment as a team.

Chapter 5 introduces the two methods used in this dissertation to automatically develop autonomous agent controllers. The first is a form of reinforcement learning that takes advantage of the hierarchical nature of the agent's task to improve the rate at which effective controllers can be learned. The second method uses grammatical evolution to evolve fuzzy rulesets used for control.

Chapter 6 details the experimental configuration used to evaluate the impacts of state and action abstraction. In addition to a description of the experimental process, a listing of the reward structure and parameters used for reinforcement learning and grammatical evolution is provided.

Chapter 7 presents the results of experiments in each of the problem domains. These results are analyzed and discussed with a particular focus on the effects of different levels of state and action abstraction.

Chapter 8 concludes this dissertation with a summary of the results, their implications, and a discussion of the potential avenues of future work.

CHAPTER 2

Related Work

The research described in this dissertation builds upon contributions found in many areas of study. As a result, familiarity with the work and perspectives found in each area is necessary to appreciate the unique aspects of the problem domain under study. First, since behavior-based architectures were designed to provide control for reactive tasks like the ones under study here, an overview of the relevant contributions from the area is presented (see Section 2.1). Next, an overview of the relevant contributions from the area of fuzzy control as it relates to action selection is provided (see Section 2.2). Lastly, an overview of some of the more relevant contributions from the two different methods of automatically generating agent controllers used in this dissertation is provided: reinforcement learning (see Section 2.3) and evolutionary computation (see Section 2.4).

2.1 Behavior-Based Robotics

A common approach to addressing the complexity of agent controllers for complex, composite tasks, is to use an individual sub-controller for each primitive task and then combine them into a complete agent controller [8]. Each of these sub-controllers, or *behaviors*, is conceptually simple and its development is intended to be straightforward. Once the individual behaviors have been developed, the overall control problem is reduced to a problem of coordinating the individual behaviors.

Pirjanian classified the different types of behavior coordination mechanisms (also referred to as action selection mechanisms) into two distinct categories (see Figure 2.1). In the first, denoted *behavior arbitration*, a single behavior is selected and given total control

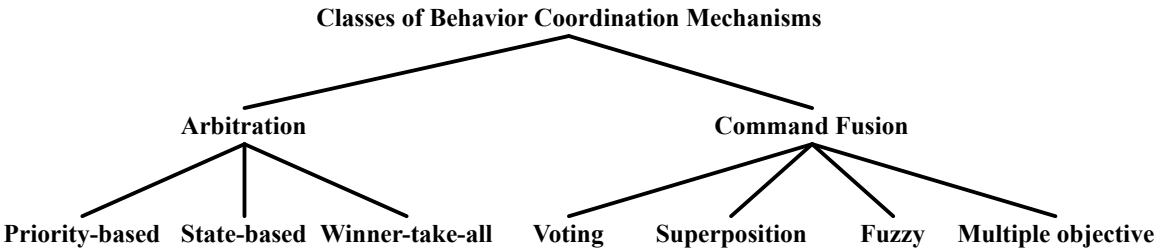


Figure 2.1: The classes of behavior coordination mechanisms as described by Pirjanian [62] are shown and redrawn here.

over the agent. Although there are a number of mechanisms to select a behavior, each one tries to select the single most relevant behavior given the agent's current state and goals. While the selection of a single behavior simplifies the later step of taking the resulting action, the use of only a single behavior limits the ability of the agent to accomplish only a single task at any given time. The subsumption architecture proposed by Brooks is an example of a behavior arbitration architecture that uses fixed priorities to select behaviors to activate [8].[1] In contrast, the second type of behavior coordination, denoted *command fusion*, allows for the activation of multiple, concurrent behaviors. In this approach, each behavior determines the action that best accomplishes its goals. Then, the individual actions are combined using a specified algorithm and an overall action is then selected. While this process is more complicated and has more potential for inappropriate actions, it does offer the agent the ability to accomplish multiple tasks, to various degrees, simultaneously. It is for this reason, that command fusion is frequently the chosen method for behavior coordination, including the work described here.

While a full discussion of all the relevant approaches found in the literature is beyond the scope of this work, a sample of some of the more relevant approaches are discussed.

Saffioti and Wasik describe a hierarchical behavior-based architecture in which a combination of behavior arbitration and command fusion, referred to as context-depending blending, is used to control robots in a RoboCup soccer game [81]. The hierarchy is developed manually and an emphasis is placed on sequential combinations of behaviors. Poten-

[1] Some implementations may not cleanly fit within this categorization.

tial conflicts between behaviors are avoided by ensuring that conflicting behaviors are never activated simultaneously. While this approach can be effective for their specific implementation, in general, the prospect of developing a controller for a command fusion architecture that ensures no conflicting behaviors are activated simultaneously is impractical.

Nicolescu and Matarić propose a hierarchical abstract behavior architecture which uses a behavior network to control a robot [57]. While it allows for concurrent behaviors, they are restricted to behaviors which are non-interfering. This is accomplished through the use of a "lock" on the robot's actuators, which a single behavior holds and thus prevents other behaviors from using.

Huber and Grupen also propose an approach that allows for concurrent, simple controllers that are equivalent to the behaviors discussed here [35]. However, there is a restriction that subordinate controllers may not prevent the primary controller from achieving its immediate objective (i.e., interference is explicitly prevented), which necessitates the identification of a primary controller. In complex, composite tasks, the development of an effective static prioritization of behaviors can be difficult and an inappropriate choice can lead to the inability of the agent to react to changes in the environment.

Pirjanian introduced a command fusion behavior coordination mechanism, referred to as multi-objective behavior coordination (or MOBC), that uses multiple-objective decision making to select actions from those recommended by behaviors [61]. While the method is applicable to the CINE tasks discussed here, there are a number of factors which make it less than ideal for our current work. First, the method for selecting the overall action is a simple, fixed heuristic. As a result, it is incapable of adapting to changes in the environment. Second, since the heuristic uses only the utilities of the recommended actions, it does not benefit from state or action abstraction. Lastly, it requires that the utilities of actions be consistent, or have the same range of values, across all behaviors. Without this consistency, one particular set of utilities could easily dominate. This approach is very similar to that of modular reinforcement learning that is further discussed in Section 2.3.2.

In a similar approach, Rosenblatt proposes utility fusion to coordinate behaviors and select actions [75, 76]. However, unlike multi-objective behavior coordination, utility fusion evaluates the utilities of potential next states, and not the actions themselves. The arbiter tasked with choosing actions chooses the action with the highest probability of reaching the next state with the highest utility. Since utility fusion is similar to multi-objective behavior coordination, the same factors which make multi-objective behavior coordination less than an ideal choice for our current work are also applicable to utility fusion.

Lastly, Behnke and Rojas describe an approach with separate sensor and reactive behavior hierarchies, similar to those used here [2]. However, unlike other approaches, including the one presented in this dissertation, each level in each of the hierarchies operate at different time frames. Furthermore, while the approach was effective, the behavior coordination mechanism was not well-defined.

2.2 Fuzzy Logic

Fuzzy logic is frequently used in behavior-based robot controllers since it allows the controller to view the state and action space as discretized while retaining the continuous nature of the values [103].[2] As a result, robot controllers using fuzzy logic are able to smoothly transition between the goals and actions of different behaviors.

One of the more common applications of fuzzy logic to robot control is in the area of command fusion (see Figure 2.1). In fuzzy command fusion, each behavior is implemented using a set of fuzzy rules whose output is the desired action in the form of a fuzzy set [62, 80]. The fuzzy set actions of each behavior are then combined into a single fuzzy set which is defuzzified to produce the overall action of the agent.

Just as in reinforcement learning (see Section 2.3.1), a hierarchical approach to fuzzy control is commonly used for complex problem domains. Raju et al. describe the construc-

[2] A more detailed description of fuzzy logic's benefits to the current problem domain can be found in Section 3.1.

tion of a fuzzy hierarchical controller in which is possible that the number of fuzzy rules used for control can be a linear function of the number of state variables as opposed to non-hierarchical controllers in which the number of fuzzy rules is exponential in the number of state variables [67]. As a result, the use of a hierarchy can prove to be a significant benefit in making the development of controllers for complex, composite tasks more practical.

Bonarini et al. discuss a multi-layer fuzzy architecture termed BRIAN in which each layer represents a different level of abstraction [7]. However, the architecture is not appropriate for our problem domains since it is not a purely reactive system. Furthermore, although the architecture has multiple layers, all the active behaviors are contained in a single layer and are not organized into a hierarchy.

Vadakkepat et al. compare a number of different fuzzy behavior-based approaches in a multi-agent RoboCup environment [96]. A significant aspect of the described approaches is that the behaviors are purely reactive and execute concurrently. However, only a high-level description of each architecture is presented and there is insufficient information to further evaluate the relevance of each to our current work.

Hoffman describes the training of a hierarchical architecture comprised of fuzzy, reactive behaviors by means of demonstration [32]. However, the architecture uses a fixed prioritization of behaviors, which, for reasons previously discussed, make it inappropriate for our uses.

Yang et al. describe a fuzzy behavior-based controller for robots that can navigate over rugged terrain [102]. A significant aspect of this work is that the controller offers theoretical guarantees of performance for the robot. However, these guarantees are only achieved through the use of Brooks' subsumption architecture and behavior arbitration, thus making it inappropriate for our needs.

Ramos et al. describe a behavior coordination mechanism using a hierarchical fuzzy architecture and demonstrate it using a goalkeeper agent in the simulated RoboCup domain [68]. Of particular interest is the abstraction of relevant state variables. Unfortunately,

behavior arbitration is used and only a single behavior is active at any given time.

Lastly, Tunstel describes a hierarchical behavior-based fuzzy control architecture that allows multiple behaviors to be active concurrently and adaptively prioritizes the behaviors depending on the agent's state [92, 93, 94, 95]. As a result, the approach is highly applicable to the types of tasks and problem domains of interest to us and is the one used in the experiments presented in this dissertation. A more in depth discussion of this architecture can be found in Chapter 3.

2.3 Reinforcement Learning

The use of reinforcement learning to learn agent controllers is well documented and the literature in the field contains many methods for simplifying the control task and improving the rate at which effective polices are learned. Unfortunately, many of these approaches make assumptions or restrictions on the tasks for which the approach can be applied. As is discussed below, these assumptions or restrictions mean they are not applicable to the CINE tasks under discussion here. While a complete review of all the approaches and the ways in which they are not applicable is beyond the scope of this work, some of the more related approaches to this dissertation are discussed below.

One of the more common restrictions in reinforcement learning approaches is that the tasks be sequential and not concurrent [49, 86, 100]. Matarić describes using reinforcement learning and behavior arbitration in a multi-agent environment [51]. Of particular interest, is the use of dense reward functions to maximize each learning opportunity and speed learning. In this dissertation, we used similar dense reward functions to not only speed learning, but also produce the desired behavior in the learned policies [50]. Despite this relevance, Matarić's approach uses sequential (i.e., non-concurrent) behaviors. A behavior is activated in response to a particular "event," and is deactivated when a new behavior is activated.

Mausam and Weld describe an approach that learns policies for multiple concurrent

tasks, but restricts the actions of these tasks to be non-interfering [53]. This means that only tasks which require non-overlapping control actions can be concurrent. Luo et al. also describe an architecture which allows for multiple, concurrent policies to be active [48]. The approach is superficially similar to the one used in this dissertation, but it does not use a hierarchy and is limited to the use of reinforcement learning. Furthermore, it does not provide any information on the exact mechanism of how actions are combined.

Provost et al. describe an approach in which reinforcement learning is used to identify useful sub-tasks in a difficult, navigation task [66]. However, as in other approaches, the focus is on episodic tasks and sub-tasks which require a sequence of actions to accomplish.

McCallum proposes a method for learning the appropriate level of abstraction of state variables based on its utility to learning [54]. These "utile distinctions" can aid in learning, much like the state and action abstraction discussed in this dissertation. However, the approach relies on a short-term memory to create these utile distinctions, and is, therefore, not a purely reactive system. As previously discussed, we are interested in only using purely reactive systems to simplify the architecture and minimize the number of potential sources which can affect learning rate and performance.

Another method of abstracting the state of an agent to simplify learning is the ignoring or removal of state variables that are deemed to be irrelevant [19, 38, 64, 37]. State variables are determined to be irrelevant if the reward for a given task, or subtask, is independent of the variable's value. For example, the state variable Y would be irrelevant if the reward function[3] depends only on X and the action taken:

$$R(\{x,y\},a,\{x',y'\}) = R(\{x\},a,\{x'\}) \tag{2.1}$$

where x and y are the initial values of state variables X and Y; a is the action taken; and x' and y' are the resulting values of the state variables X and Y. While the removal of irrelevant

[3] A *reward function* defines the intrinsic value of a state in a reinforcement learning problem and is used to identify the good and bad events [87].

variables is an effective method of state abstraction, automated methods of identifying irrelevant variables were not used in the work presented here for two reasons. First, it is entirely possible that in the tasks and problem domains used, none of the variables would be determined to be irrelevant. As a result, no state abstraction would be performed. For example, in the tasks used in this dissertation, all the state variables are relevant to their associated primitive tasks. As such, no variables would be classified as irrelevant and the state would not be able to be abstracted. Second, we wanted to minimize the number of potential sources which could affect the learning rate and performance, so we used a manual approach that could be made to be consistent across all tasks and problem domains used. Since the approach that was used was manual, we had direct control over the abstraction process. As a result, over-abstraction of the agent's state, and the associated deleterious effects, was possible (see Section 6.2 for a more complete discussion).

2.3.1 Hierarchical Reinforcement Learning

A number of reinforcement learning approaches exist which leverage the hierarchical nature of many problem domains. These approaches focus on learning policies for the hierarchical decomposition of a given task. Despite the variety of approaches, most of these approaches, just like the other reinforcement learning approaches discussed above, are not applicable to the CINE tasks under study in this work because of specific assumptions or restrictions of the approach.[4]

The learning of macro actions consisting of a sequence of actions, commonly referred to as *options*, is frequently used to provide abstraction and speed learning of a complex task [65, 88]. However, options are not compatible with CINE tasks for a number of reasons. First, the process of learning options is designed to be useful in episodic tasks since the learning process depends on a task having termination criteria. Second, options are primarily used in tasks where only a single task is active at any given time. While there

[4]For a more detailed comparison of many of the approaches discussed here, the reader is directed to Barto and Mahadevan [1].

has been work on architectures in which multiple options can be active concurrently, they too have restrictions. Early work restricts the actions taken by concurrent options to be non-interfering [73, 74]. More recent work has lifted this restriction, but is only able to achieve this by an a priori ranking of subgoals, which are still assumed to be episodic [72].

The MAXQ hierarchical reinforcement learning algorithm assumes that a hierarchical decomposition of a task is given and attempts to learn the policies at each level simultaneously [18, 19]. Just as in the work presented in this dissertation, a significant aspect of the MAXQ algorithm is its focus on the abstraction of state variables. However, the MAXQ algorithm is only applicable to episodic tasks since it assumes the existence of termination criteria. Unlike MAXQ, the HEXQ reinforcement learning algorithm does not require an a priori decomposition of a task [30]. It is capable of learning both the hierarchical decomposition and the policies of each subtask. However, it too assumes that subtasks are sequential and finite horizon, or episodic. Layered learning is another example of a hierarchical learning algorithm [85]. It promotes the reuse of previously developed policies in new tasks, but also concentrates on sequential, episodic primitive tasks. Lastly, Ghavamzadeh et al. developed a hierarchical reinforcement learning algorithm primarily for use in multi-agent environments, but it too assumes episodic and sequential subtasks [26].

In an approach similar to the one presented in this dissertation, Bonarini describes using reinforcement learning to learn individual primitive behaviors and how to best combine them [6]. Like our work, functional behavior decomposition is used to decompose the overall behavior into separate behaviors which have distinct state variables and goals. While similar in theme, however, the architecture used does not allow for concurrent behaviors.

Other hierarchical reinforcement learning approaches exist which do not explicitly restrict the number of active subtasks, but have other attributes which make them inapplicable. For example, the approaches of Parr and Russell [59] and Singh and Cohn [82] do not restrict the number of active tasks, but instead restrict the actions chosen. In the latter case, the selection of an action for one subtask, restricts the selection of an action for

another subtask. Lastly, Koller and Parr use reinforcement learning with tasks comprised of multiple subtasks and do not explicitly restrict the number of active subtasks [41]. Unfortunately, their work focuses only on automatically decomposing the value function of a policy for the overall task and not on learning an effective policy.

2.3.2 Modular Reinforcement Learning

Humphrys [36] and Karlsson [40] independently describe a reinforcement learning algorithm that is appropriate for the CINE types of problems under study here. In this algorithm, commonly referred to as *modular* reinforcement learning [4, 84], a policy for each active primitive task is learned simultaneously using the state information and rewards local only to the task. At each time step, the policy for each subtask provides the action selection mechanism with a utility value for each possible action. This utility value is calculated using the value of taking a particular action from a given state, referred to as a *Q-value*, and is often simply the Q-value itself. These utilities are then used by the action selection mechanism to choose the action that the agent will take. The approach used in this dissertation to choose the action, called the "greatest mass," simply chooses the action with the highest utility, or sum of Q-values, across all the primitive task policies [40]:

$$Q_{gm}(s,a) = \sum_{i=1}^{N} Q_i(s_i,a) \tag{2.2}$$

where Q_{gm} denotes the greatest mass utility of a state-action pair, s_i is the current state of the environment relevant to primitive task i, a is an action available to the agent, and Q_i is the Q-value in primitive task i of taking action a in state s_i. To update each policy, the total reward must be decomposed into rewards specific to each primitive task. Constructing a reward function for a composite task can be deceptively difficult, for even a good reward function can produce undesirable behaviors [50]. A common method is to simply use the sum of the reward functions of each primitive task as the reward function for the composite

task:

$$R(s,a,s') = \sum_{i}^{N} R_i(s_i,a,s_i'). \tag{2.3}$$

where s_i is the agent's state in primitive task i, s is the agent's state in the overall composite task, a is the action taken, s_i' is the resulting state in primitive task i, s' is the resulting state in the overall composite task, and R_i is the reward earned in primitive task i. However, this method makes the implicit assumption that rewards are consistent across all the primitive tasks which complicates the process of learning the policies for the primitive tasks [4]. While the construction of the composite task's reward function is designed to promote specific traits in the composite task's policy (e.g., a risk-averse policy versus a risk-taking policy), the unintended consequence is that the policies of the primitive tasks show the effects of these traits. This is due to the fact that, in modular reinforcement learning, all learning takes place in the policies of the primitive tasks. As a result, the policy for a given primitive task could potentially only be useful in the composite task for which it was originally learned.

Q-learning cannot be used in learning the primitive task policies since it is off-policy and assumes the optimal policy will be followed [98]. One cannot assume that the optimal policy will be followed for modular reinforcement learning since primitive task policies must share control of the agent [78, 84]. As a result, an on-policy learning method, such as the Sarsa algorithm, must be used [77]. In Sarsa, updates are made using only the policy currently being followed using the sequence of events $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, and is written as follows:

$$Q_i(s_i,a) \leftarrow (1-\alpha)Q_i(s_i,a) + \alpha(r_i + \gamma Q_i(s_i',a')). \tag{2.4}$$

As can be observed, Sarsa uses the Q-value of the state-value pair that was actually observed. As a result, the Q-values in a policy for a given primitive task reflect the rewards received while operating in conjunction with the policies of other primitive tasks.

A further complication in the use of modular reinforcement learning, as it is currently

implemented, is that it requires that the policies for primitive tasks provide the learned Q-value for a given action. Therefore, it is unable to reuse polices developed using methods other than reinforcement learning. While it is possible, in general, to learn the Q-values for an existing policy offline, this method can produce the same bias in the Q-values that the use of Q-learning produces since the Q-values must reflect the shared control of the agent. Even though alternative methods could be used to provide utility values without the use of Q-values [63], modular reinforcement learning still depends on control decisions flowing up from the low-level policies. As a result, there is no scaling advantage to extending beyond a two-level hierarchy where learning occurs at the lowest level since the top level merely uses a simple heuristic to combine the lower-level results.

2.3.3 Transfer Learning

As previously discussed, the ability to reuse controllers designed for one task in another can provide significant benefits in the development of effective agent controllers. In reinforcement learning, this is referred to as *transfer learning* and is an area of study that is gaining increasing attention. Unfortunately, most of the approaches are not applicable to the tasks and problem domains used in this work. In one of the more promising approaches, Taylor et al. propose a method of mapping action-value functions between similar tasks [91]. Unfortunately, this approach is not applicable since it uses only a single policy and relies on the use of reinforcement learning and the existence of an action-value function. Konidaris and Barto propose a method of using transfer learning with options [42], but, as previously discussed, our tasks and problem domains do not benefit from options. Fernández and Veloso propose a method that uses existing policies to learn a policy for a similar task [25]. However, the environments used were simple and used only a single task. Furthermore, the new tasks were very similar to old tasks and only differed in the initial conditions. Talvitie and Singh propose a method for using existing candidate policies to learn a policy for a single task [89]. In this method, all the candidate policies are for the same task and the

learned policy chooses which candidate policy to use in each state.

2.4 Evolutionary Computation

Another approach found in the literature to automatically developing agent controllers is the use of evolutionary computation. Historically, genetic algorithms [33] or genetic programming [43] are the primary approaches used to evolve controllers.

Of particular interest in the work presented in this dissertation is the combination of evolutionary computing and fuzzy control.[5] While there are a number of examples of this combination [31], only a sample are described here.

Evolutionary computation is commonly used to evolve fuzzy controllers in two ways. In the first way, evolution is used to evolve the membership functions of linguistic values so as to effectively tune a fuzzy controller's existing fuzzy rules. In the second way, evolution is used to evolve fuzzy rules that make use of existing membership functions. The evolution of fuzzy rules can either consist of a population of individual rules that are combined to build a single set of rules (referred to as the "Michigan" approach) or a population of rule sets that are individually used for control (referred to as the "Pitt" approach) [15].

Bonarini uses genetic algorithms and learning classifier systems to identify some of the difficulties in evolving fuzzy rules for fuzzy controllers [5]. A significant aspect of this work is a discussion that the "Michigan" approach can cause detrimental competition between fuzzy rules. Furthermore, the authors propose an alternative method, termed *ELF*, that seeks to address problems found in both the "Michigan" and "Pitt" approaches. However, it is only applicable to environments with relatively few states.

Chen et al. use evolutionary programming to evolve hierarchical fuzzy systems [9]. The hierarchy and rulesets are evolved in a succession of iterations until satisfactory performance is achieved. Unfortunately, this approach is not used for control and uses the

[5]The area of study focusing on the combination of evolutionary computing and fuzzy control is commonly referred to as *soft computing* or *computational intelligence*.

Takagi-Sugeno type fuzzy system where a fuzzy rule's consequence is a function mapping input space directly to output space. This approach is not as intuitive as the approach used in this dissertation and requires a derivation of nonlinear system equations from membership functions.

Vadakkepat et al. describes a method that combines the two ways of evolving fuzzy controllers which coordinate behaviors [97]. In this approach, evolution is used to evolve both the fuzzy rules and the membership functions. While the organization of the architecture appears to be similar to the approach used in this dissertation, there are many implementation details missing from the description and, therefore, prevent a complete comparison. Other approaches also evolve both the fuzzy rules and membership functions, usually in a process that iteratively refines each until a performance threshold is reached [52, 10].

CHAPTER 3

Adaptive Fuzzy Behavior Hierarchies

In ordinary, or *crisp*, set theory, an instance either belongs to a set or it does not. In contrast, in fuzzy set theory a value can have partial membership in a set.[1] For example, the temperature in a room can be somewhere in between fully `cold` and fully `cool`. This distinction can provide many benefits for the frequent case in control where a continuous variable must be discretized into a finite set of values. For example, consider the case of designing a controller for a heater. Creating an effective controller would be simplified greatly if the input (i.e., the temperature) and the output (i.e., the heater's power level) could be discretized into a few specific values. A possible discretization using crisp set theory is shown in Figure 3.1. While such a discretization may seem reasonable, the crisp discretization means that the resulting control surface of the controller is discontinuous and can result in control actions which rapidly oscillate between two very different actions. Such oscillations are only exacerbated by the relatively coarse granularity of the discretization shown here. For example, if the following control rules were used with crisp values, the control surface in Figure 3.2 would be the result:

IF *temperature* is `cold` THEN *heater* is `high`

IF *temperature* is `cool` THEN *heater* is `low`

IF *temperature* is `warm` THEN *heater* is `off`

On the other hand, if fuzzy set theory is used for discretization, such as the one shown in Figure 3.3, the control surface would provide smooth transitions between each "discrete"

[1] A full explanation of fuzzy logic and control is beyond the scope of this work. The reader is directed to Lee [44, 45] for a short overview or Driankov et. al [20] for a more in depth discussion.

Figure 3.1: The crisp variables *temperature* and *heater* are used by a controller for a heater. The *temperature* crisp variable is an input value to the controller and has three values: `cold`, `cool`, and `warm`. The *heater* crisp variable is an output value from the controller and denotes the desired power level of the heater. It has three values: `off`, `low`, and `high`.

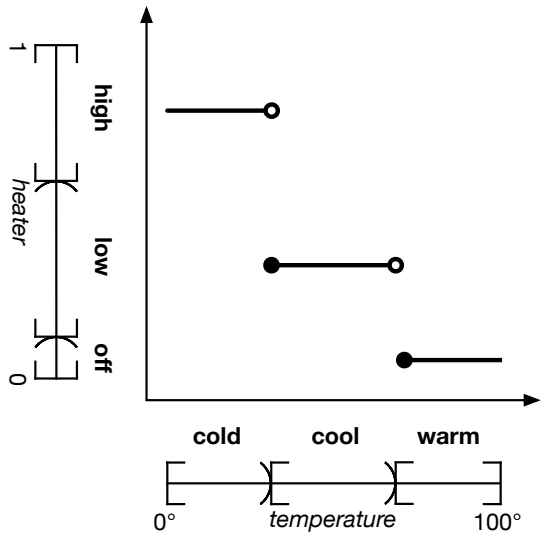


Figure 3.2: A control surface produced from control rules using crisp sets is shown. Note that the interval used in the *heater* crisp set is reversed to improve readability of the control surface.

value (see Figure 3.4). This distinction can provide significant benefits in agent control and is why fuzzy control is used in the work presented here.

3.1 Fuzzy Control

As was previously mentioned, in crisp set theory a variable's value either belongs to a particular set, or it does not. For example, if the discretization described in Figure 3.1 were to be used, a temperature t is either `cold` or it is not. Another way of describing this is to say t has a membership in `cold` of either 1 or 0. If we only consider the range of temperatures from 0° to 100°, referred as the temperature's *universe of discourse*, we could

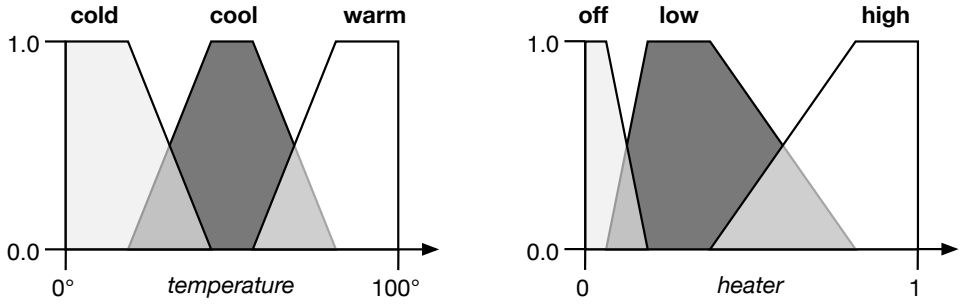


Figure 3.3: The linguistic variables *temperature* and *heater* are used by a fuzzy controller for a heater. The *temperature* linguistic variable is an input value to the fuzzy controller and has three linguistic values: `cold`, `cool`, and `warm`. The *heater* linguistic variable is an output value from the fuzzy controller and denotes the desired power level of the heater. It has three linguistic values: `off`, `low`, and `high`.

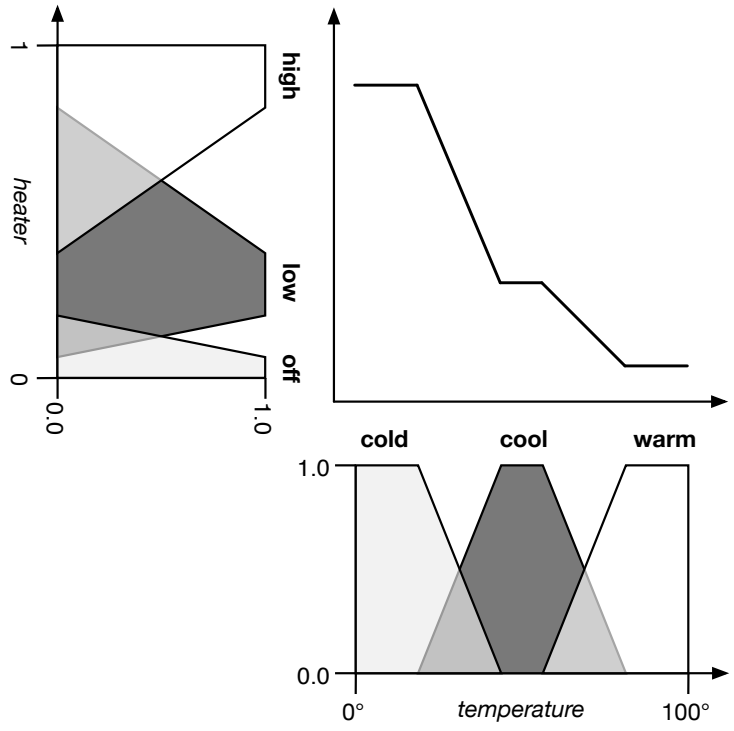


Figure 3.4: A control surface produced from control rules using fuzzy sets is shown. Note that the interval used in the *heater* fuzzy set is reversed to improve readability of the control surface.

formalize the *membership functions* for calculating an arbitrary temperatures membership in each set as follows:

$$cold_C(t) = \begin{cases} 1, & \text{for } t \in [0,30) \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

$$cool_C(t) = \begin{cases} 1, & \text{for } t \in [30,70) \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

$$warm_C(t) = \begin{cases} 1, & \text{for } t \in [70,100] \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

However, if fuzzy discretizations are used, a temperature could be both `cold` and `cool` with varying degrees. Another way of describing this is to say that t has a membership in `cold` in the interval $[0,1]$. If we consider the same universe of discourse as above, the membership functions in the fuzzy case could be as follows:

$$cold_F(t) = \begin{cases} 1, & \text{for } t \in [0,20] \\ (40-t)/20, & \text{for } t \in (20,40) \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

$$cool_F(t) = \begin{cases} (t-20)/20, & \text{for } t \in (20,40) \\ 1, & \text{for } t \in [40,60] \\ (80-t)/20, & \text{for } t \in (60,80) \\ 0, & \text{otherwise} \end{cases} \tag{3.5}$$

$$warm_F(t) = \begin{cases} (t-60)/20, & \text{for } t \in (60,80) \\ 1, & \text{for } t \in [80,100] \\ 0, & \text{otherwise} \end{cases} \tag{3.6}$$

Using the fuzzy membership functions, a *temperature* of 25° has a membership of 0.75 in `cold`, 0.25 in `cool`, and 0 in `warm`. Using the crisp membership functions, the same temperature would have a membership of 1 in `cold` and memberships of 0 in both `cool` and `warm`.

In this example, both *temperature* and *heater* are examples of *linguistic variables*. Linguistic variables represent variables used by the controller either as input or output. A linguistic variable is associated with a finite set of *linguistic values*, each of which corresponds to a different discretized value of the variable.[2] In the current example, the linguistic variable *temperature* has three different linguistic values: `cold`, `cool`, and `warm`.

Since input values to the controller are real-valued numbers, they must be converted to fuzzy numbers using a process called fuzzification. In fuzzification, the crisp value of a linguistic variable is converted to a fuzzy set of membership values for each of the linguistic values associated with the variable. The range of potential values of a linguistic variables is referred to as the variable's *universe of discourse* and is denoted U. A membership function μ is used to assign every $u \in U$ a membership value in the interval $[0,1]$ for a given fuzzy set:

$$\mu_F : U \rightarrow [0,1]. \tag{3.7}$$

Note that membership in the fuzzy set is in the *interval* $[0,1]$ and not from the set $\{0,1\}$ as in crisp sets. Formally, a fuzzy set F in a universe of discourse U is defined by [20]:

$$F = \{(u, \mu_F(u)) \,|\, u \in U\} \tag{3.8}$$

[2]The linguistic values and the associated membership functions for the linguistic variables used in this work are discussed in detail in Appendix E.

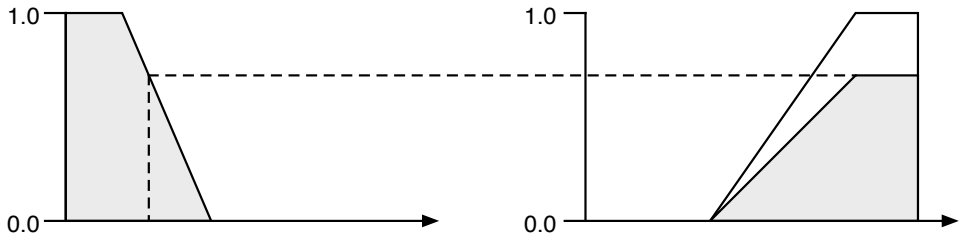


Figure 3.5: Larsen implication operator

where μ_F is a membership function for the fuzzy set F. In the case where the membership functions are continuous, the fuzzy set F can be calculated as follows [44]:

$$F = \int_U \frac{\mu_F(u)}{u} \, \mathrm{d}u \tag{3.9}$$

where u denotes the linguistic variable's value. In the case where the membership functions are discrete, such as the ones described in the current example, and $U = u_1, \cdots, u_n$, F can be calculated using the following:

$$F = \sum_{i=1}^{n} \frac{\mu_F(u_i)}{u_i} \tag{3.10}$$

where u_i denotes one of n discrete values within the universe of discourse.

As in crisp set theory, there are a number of operations that can be performed on fuzzy sets. Some of the more notable operations are the intersection, union, and sum. While a more in depth discussion of these operations is beyond the current scope,[3] a fundamental property of these operations is that they operate on fuzzy sets and produce another fuzzy set.

Fuzzy controllers are usually implemented using fuzzy rules. A fuzzy rule uses one or more input values to produce a fuzzy set for an output value, which will later be converted to a crisp value, commonly referred to as *defuzzification*. A basic fuzzy rule could have the form:

[3]The reader is directed to [20] for an in depth discussion of these operations.

IF x is $\tilde{A}$ and y is $\tilde{B}$ THEN z is $\tilde{C}$

where x and y are input values to the controller; z is an output value of the controller; and $\tilde{A}$, $\tilde{B}$, and $\tilde{C}$ are linguistic values (which equate to fuzzy sets) corresponding to the linguistic variables of x, y, and z respectively. The fuzzy rule is comprised of two main parts: an antecedent and a consequent. The antecedent consists of a proposition which evaluates and returns an input value's membership in a given linguistic value. A sample proposition using the heater controller example would be "*temperature t* is `cold`." The fuzzy rule shown above uses a compound proposition that is built using a conjunction of basic propositions. In such situations, the resulting membership is calculated using the intersection, or t-norm, as follows:

$$a \wedge b = \min(a, b) \tag{3.11}$$

where a and b are membership values as computed by individual propositions. An example where the intersection would be useful is if we added a second state variable representing the time of day, denoted *time-of-day*, to the temperature controller. We could then create fuzzy rules such as the following:

IF *temperature* is `cold` and *time-of-day* is `night` THEN *heater* is `high`

If the membership in `cold` were 0.75 and the membership in `night` were 0.8 then the membership of the antecedent would be $\min(0.75, 0.8) = 0.75$. The consequent in a fuzzy rule uses the membership value computed by the antecedent to create a fuzzy set. A sample consequent using the heater controller example would be the "*heater* is `high`" found in the previous rule. The operator used in this work to compute the resulting fuzzy set is the Larsen implication operator and is defined as follows:

$$\mu_{FR} = \mu_I \mu_O \tag{3.12}$$

where μ_I is the membership of the input values as computed by the antecedent, μ_O is the membership of the linguistic value in the consequent, and μ_{FR} is the membership produced

by the fuzzy rule. This particular implication operator has a scaling effect on the membership of the output value (see Figure 3.5). Potential fuzzy rules for the heater example are shown in Figure 3.6.

When fuzzy rules are combined to form a single controller, the result is called a ruleset and could have the following form:

$$\begin{aligned} R_1\text{:} &\quad \text{IF } x \text{ is } A_1 \text{ and } y \text{ is } B_1 \text{ THEN } z \text{ is } C_1, \\ R_2\text{:} &\quad \text{IF } x \text{ is } A_2 \text{ and } y \text{ is } B_2 \text{ THEN } z \text{ is } C_2, \\ &\quad \ldots \\ R_M\text{:} &\quad \text{IF } x \text{ is } A_M \text{ and } y \text{ is } B_M \text{ THEN } z \text{ is } C_M \end{aligned}$$

Since the result of a rule is a fuzzy set, each of the fuzzy sets produced by the rules in a ruleset can be combined into a single fuzzy set using a fuzzy union operator as follows:

$$\mu_{FRS} = \bigcup_{i=1}^{M} \mu_i \tag{3.13}$$

where μ_{FRS} is the fuzzy set produced by the entire fuzzy rule set and μ_i is the fuzzy set produced by rule i. Figure 3.6 depicts the process of combining the fuzzy sets of each rule in the ruleset to produce a fuzzy set which can be defuzzified to find the crisp output value used for control. This process of combining each fuzzy set is what gives fuzzy controllers the ability to smoothly transition between different control actions.

One difficulty in using fuzzy rules for control is the number of rules required to completely cover the state space. If there are n state (or input) variables each with m linguistic values (or fuzzy sets), then a *complete* set of fuzzy rules has m^n rules. While it is impractical to have a complete ruleset for every controller, it is still apparent that an effective controller that uses fuzzy rules could require a fuzzy rule set that is exponential with respect to the number of input values. Therefore, an approach is needed to combat this complexity. One possible approach is the use of a hierarchy fuzzy rulesets. Previous work has shown that the use of a hierarchy can reduce the number of rules for a hierarchy to be a linear function of the input values [67].

After the input has been fuzzified and the fuzzy controller has performed the necessary

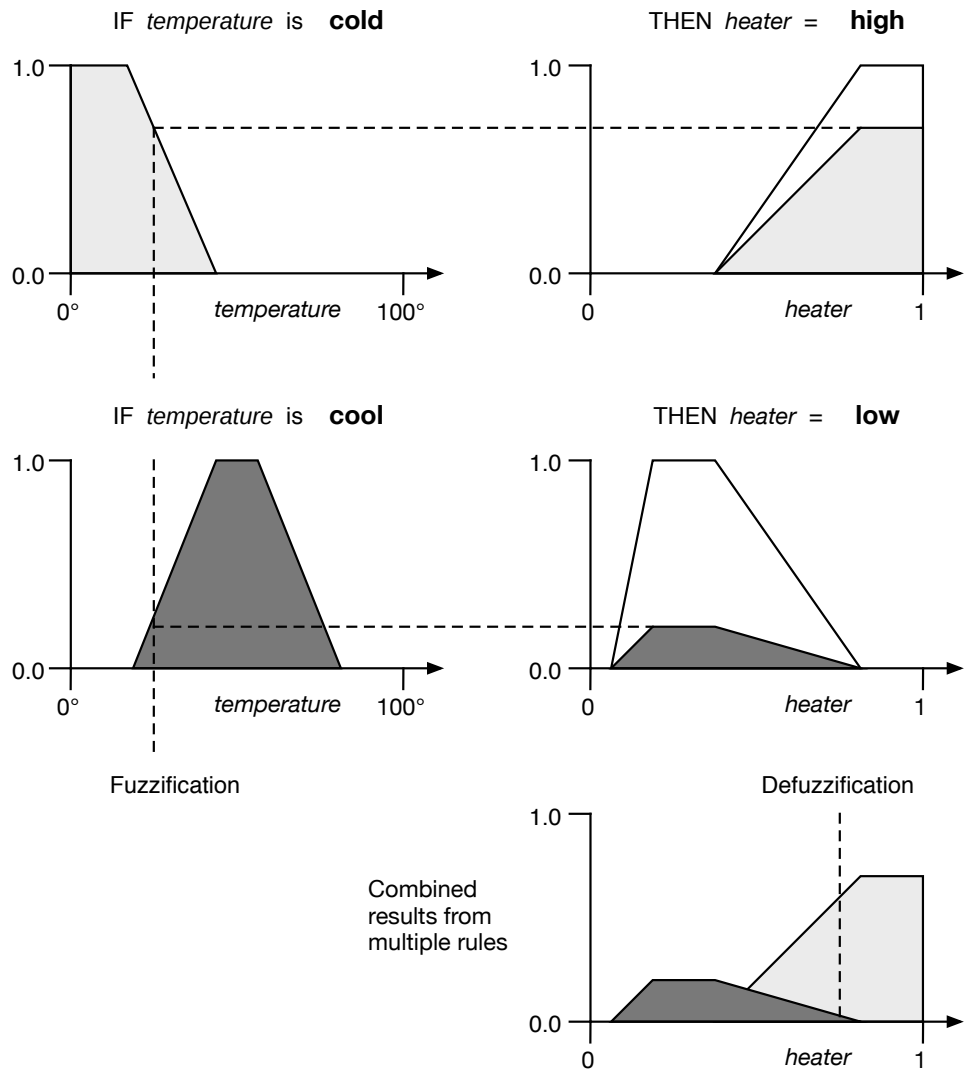


Figure 3.6: Two fuzzy rules are shown contributing to the overall power level of the heater. Since the *temperature* variable is fuzzy, more than one rule applies to the current state, albeit with different levels of activation. The output of each rule is weighted by the membership of the *temperature* to the linguistic value of the antecedent.

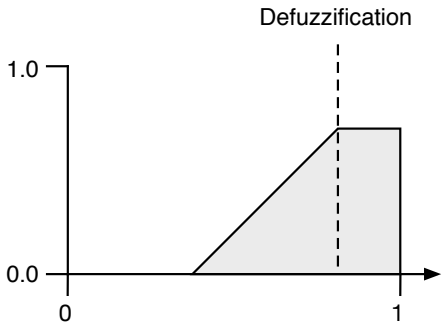


Figure 3.7: Simple defuzzification can be viewed as computing the center of the fuzzy set's area

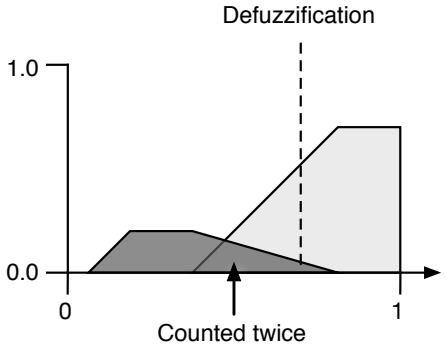


Figure 3.8: Center-of-Sums defuzzification

calculations, a fuzzy set is produced. This fuzzy set is usually produced using a combination of the previously mentioned operators and is itself built using other fuzzy sets. To be useful in control, the output fuzzy set must be converted to a real-valued number before it can be acted upon. This conversion process is called defuzzification. In defuzzification, the membership values of each linguistic value is used in conjunction with the linguistic value's membership function to produce a crisp number. In a graphical sense, a general defuzzification process can be viewed as computing the center of the fuzzy set's area (see Figure 3.7). There are a wide array of different approaches to defuzzification [20] and their full discussion is beyond the scope of this work. However, the approach used in this work, Center-of-sums, merits discussion since it will be used later. A distinguishing aspect of the Center-of-sums defuzzification process is that it takes the sum of each of the individual fuzzy set's membership values instead of the union (see Figure 3.8). As a result, areas of overlap are counted more than once. This is an example of *weight counting* and is important to the controller architecture discussed below. In the continuous case, the Center-of-sums is computed as follows:

$$u^* = \frac{\int\limits_{u \in U} u \sum\limits_{k=1}^{\ell} \mu_k(u)\,\mathrm{d}u}{\int\limits_{u \in U} \sum\limits_{k=1}^{\ell} \mu_k(u)\,\mathrm{d}u} \tag{3.14}$$

where u^* is the real-valued, defuzzified number and μ_k is one of ℓ fuzzy sets contributing to the overall fuzzy set. In the discrete case, it is computed by the following:

$$u^* = \frac{\sum\limits_{i=1}^{n} u_i \sum\limits_{k=1}^{\ell} \mu_k(u_i)}{\sum\limits_{i=1}^{n} \sum\limits_{k=1}^{\ell} \mu_k(u_i)} \tag{3.15}$$

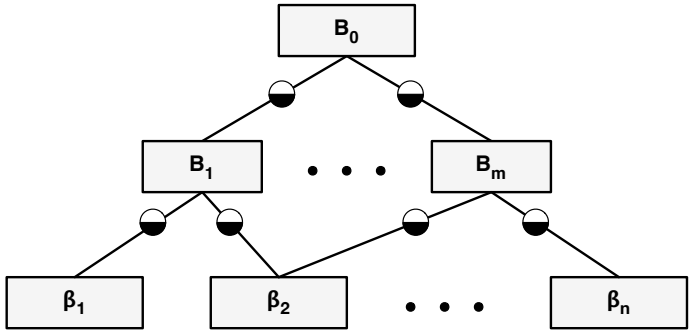


Figure 3.9: A set of primitive behaviors (denoted β_i) are organized into a hierarchy and adaptively weighted by composite behaviors (denoted B_m) as described by Tunstel [93] and redrawn here. The half-filled circles denote the weights and threshold values used to modulate behaviors.

3.2 Adaptive Fuzzy Behavior Hierarchies

An example of a hierarchical, fuzzy approach to agent control, and the one used for these experiments, is an adaptive fuzzy behavior hierarchy [93]. The hierarchy is organized using two types of behaviors. Behaviors responsible for accomplishing simple, primitive tasks are called *primitive behaviors* (see Figure 3.9). Primitive behaviors reside at the lowest level of the hierarchy and are responsible for producing low-level control actions for the agent. Since each primitive behavior is responsible for a single primitive task, inputs to the behavior consist only of state information relevant to the associated primitive task.

Following the formulation of Tunstel, let X and U be the sets of all possible input and output values, or universes of discourse, for a primitive behavior with a ruleset of size M. Individual rules within the ruleset have the following form:

$$\text{IF } x \text{ is } \tilde{A}_i \text{ THEN } u \text{ is } \tilde{B}_i \tag{3.16}$$

where x represents the linguistic variables describing primitive task state information, such as direction or distance, and u represents linguistic variables describing motor command actions, such as steering direction and speed, $\tilde{A}_i$ and $\tilde{B}_i$ represent the fuzzy linguistic values corresponding to the variables x and u. The antecedent proposition "x is $\tilde{A}_i$" can be

replaced with a compound antecedent using a conjunction or disjunction of propositions. The consequent "u is $\tilde{B}_i$" could also be replaced with a compound consequent. For example, a primitive behavior responsible for steering towards a goal could have the following rule:

IF *goalDir* is `LEFT` THEN *steerDir* is `LEFT`

The output of the i-th fuzzy rule is formally defined as:

$$\tilde{u}_i \in X \times U \tag{3.17}$$

which is a fuzzy set. As noted above in Equation 3.13, the output of the entire ruleset of M rules for a primitive behavior p is also a fuzzy set, denoted $\tilde{\beta}_p$, and can be determined by finding the union of the fuzzy sets from each of the M rules:

$$\tilde{\beta}_p = \bigcup_{i=1}^{M} \tilde{u}_i \tag{3.18}$$

If a fuzzy controller consists of only a single, active primitive behavior, $\tilde{\beta}_p$ could be defuzzified to produce low-level control commands.

The output fuzzy set of each primitive behavior can be combined in a similar manner to produce a single output. However, since primitive behaviors often have conflicting goals, their actions often conflict as well. A method of assigning different activation levels to different primitive behaviors could address these conflicts and allow an agent to accomplish its overall composite task. In an adaptive fuzzy behavior hierarchy, this is accomplished by means of *behavior modulation* in which the activation levels of primitive behaviors are adjusted, or adapted, based on the current overall state of the agent. These activation levels are referred to as degrees of applicability (DOA) and are assigned to primitive behaviors by a high-level, *composite behavior*. Composite behaviors are only responsible for modulating other behaviors, either primitive or composite, and do not produce low-level

control commands. For example, a composite behavior that is responsible for modulating a COLLISIONAVOIDANCE primitive behavior and a GOALSEEK primitive behavior could determine that since a collision is not imminent, the GOALSEEK behavior is more applicable and should have a `HIGH` activation, while the COLLISIONAVOIDANCE behavior should have a `LOW` activation. Composite behaviors are also implemented using fuzzy rulesets, but, since they produce outputs specifying activation levels and use different output fuzzy linguistic variables and values, their consequents differ from those found in primitive behaviors. Fuzzy rules within a composite behavior have the basic form:

$$\text{IF } x \text{ is } \tilde{A}_i \text{ THEN } \alpha \text{ is } \tilde{D}_i \tag{3.19}$$

where $\tilde{A}_i$ is the same as that defined in Equation 3.16, α is the scalar activation level of a given behavior, and $\tilde{D}_i$ represents the fuzzy linguistic values (e.g. `LOW`, `MEDIUM`, `HIGH`) corresponding to the activation levels which are used to modulate a behavior. If a behavior is not explicitly given an activation level, it is automatically given a default activation of 0 and does not contribute to the overall output of the controller. Furthermore, threshold values can be used to provide cutoff points for a modulated behavior's activation [92]. Just as with primitive behaviors, the output of a composite behavior is a fuzzy set. However, when defuzzified, the crisp values provide the current activation levels of lower-level behaviors, and not motor control commands. Using fuzzy rulesets to produce activation levels results in smooth transitions between different sets of activation levels in response to the changing state of the agent.

The activation level α_p of a modulated behavior p is used to calculate the weighted contribution of the behavior to the overall controller's output using the following:

$$\alpha_p \cdot \tilde{\beta}_p \tag{3.20}$$

where $\tilde{\beta}_p$ is the output of the behavior, as defined in Equation 3.18. The output of each

primitive behavior can now be combined using their respective activation levels to weight their overall contribution to the action generated by the controller. The output of the entire behavior hierarchy is calculated as follows:

$$\tilde{\beta}_H = \biguplus_{p \in P} \alpha_p \cdot \tilde{\beta}_p \tag{3.21}$$

where $\tilde{\beta}_H$ is the output of the entire behavior hierarchy, P is the set of all primitive behaviors, and $\biguplus$ is the arithmetic sum of the fuzzy sets over all the primitive behaviors. The fuzzy output values are then defuzzified using the discrete form of Center-of-Sums defuzzification as follows [20][4]:

$$u^* = \frac{\sum_{i=1}^{n} u_i \sum_{p \in P} \mu_{\tilde{\beta}_H}(u_i)}{\sum_{i=1}^{n} \sum_{p \in P} \mu_{\tilde{\beta}_H}(u_i)} \tag{3.22}$$

where u is the motor command output fuzzy variable and μ is the membership function defined over the set of all possible actions.

Since composite behaviors only modulate lower-level behaviors using state information, composite behaviors do not require lower-level behaviors to provide any information to aid in the modulation process. This is in contrast to other behavior coordination mechanisms which, for example, may require low-level behaviors to indicate the utility of a specific action [63]. The only restriction that an adaptive fuzzy behavior hierarchy places on modulated behaviors is that primitive behaviors produce a fuzzy set as output since fuzzy inferencing is used to combine their outputs into a single action. As a result, although the hierarchy is designed to use behaviors built using fuzzy rulesets, alternative methods can be used to implement behaviors as long as they produce a fuzzy set. We will later use this to our advantage when we apply machine learning techniques to the process of automatically

[4]Much of the literature in the field, including the Tunstel's work [93], cite the continuous form of the Center-of-Sums defuzzifier.

developing behaviors.

It is important to note that since a composite behavior does not produce low-level control actions, it may not need the full joint state space of the composite task to provide effective behavior modulation. For example, it is possible that the direction of the closest collision is irrelevant when determining the modulation for a COLLISIONAVOIDANCE primitive behavior. It may be that only the estimated time until the collision is important. As a result, it may be possible to reduce the state information used by the modulation process in a composite behavior that provides comparable performance. A reduced state set such as this would provide significant benefits not only in reducing the complexity of the composite behavior's ruleset, but also in the effort required to develop the ruleset itself. This has significant implications for improving the practicality of developing controllers for the complex composite tasks under study.

There are two ways in which the state space can be reduced. First, state information that is determined to be irrelevant can simply be removed from the state space in a process commonly referred to as subset feature selection. As previously discussed, there are many methods for finding irrelevant features [28, 17, 69, 101], but they assume that irrelevant features exist within the state space. Such an assumption may not be valid in the tasks used in this thesis. Furthermore, to better evaluate the effects of state abstraction, we wished to use a method that offered more fine control of the abstraction process. In the second option, state information is converted into a more abstract form. The process of finding these abstractions, known as feature extraction [29], can result in either fewer state variables or no change in the number of state variables. However, the extracted variable set is simpler than the original variable set. For example, a composite behavior may not need to know the exact relative direction to an object and only requires the magnitude of the direction for effective control. In this example, the original *direction* state variable is extracted to a more simple representation where both `SMALL_LEFT` and `SMALL_RIGHT` are abstracted to the same `SMALL` value. Since primitive tasks are, by definition, simple

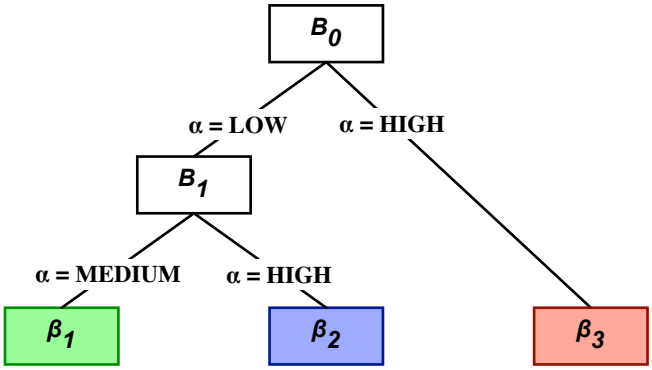
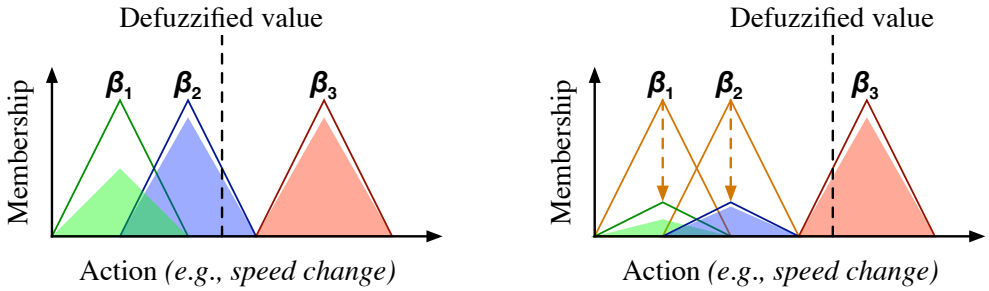


Figure 3.10: A sample three-level hierarchy.



(a) Defuzzification in original

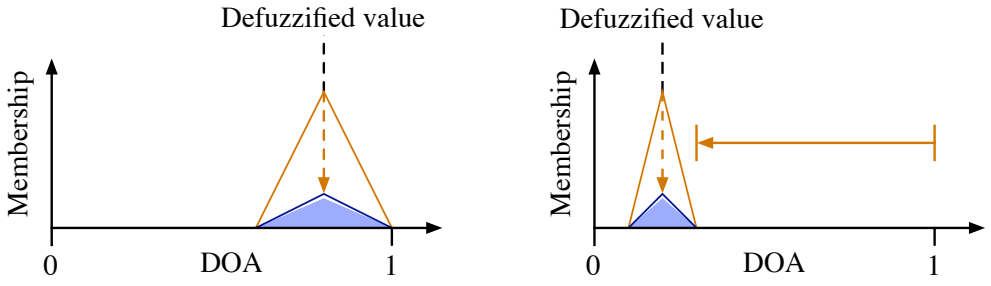
(b) Defuzzification in extension

Figure 3.11: A comparison of modulation effects on primitive behaviors using the original implementation and our extension are shown. The effects of behavior modulation, or weighting, in the original implementation of adaptive fuzzy behavior hierarchies are shown on the left. The effects of behavior modulation using our extension are shown on the right and are more like one would expect to see.

and straightforward, one can easily determine abstractions that may be beneficial. As a result, this is the method that will be used in this work (see Section 6.2).

3.3 Extending Adaptive Fuzzy Behavior Hierarchies

Although adaptive fuzzy behavior hierarchies have been shown to provide effective control, their implementation, as described by Tunstel, limits their application to two-level hierarchies. To illustrate this limitation, consider the three-level behavior hierarchy in Figure 3.10. The primitive behaviors β_1 and β_2 are coordinated by the composite behavior B_1. The composite behavior B_1 and another primitive behavior β_3 are then coordinated by



(a) DOA calculation in original (b) DOA calculation in extension

Figure 3.12: A comparison of activation level, or DOA, calculation using the original implementation and our extension are shown.

the top-level composite behavior B_0. If B_0 chooses to weight B_1 with a `LOW` weight and weight β_3 with a `HIGH` weight, one would expect that the primitive behaviors β_1 and β_2, regardless of the weights assigned by B_1, would have much less influence over the resulting action than the primitive behavior β_3, because their parent behavior, B_1, was given a `LOW` weight (e.g., see Figure 3.11b). However, this is not the case. The overall contribution of the primitive behaviors β_1 and β_2 depends *only* on the weights assigned by B_1, which are not affected by B_1's `LOW` weighting (see Figure 3.11a).

One reason adaptive fuzzy behavior hierarchies are able to produce effective control actions is that the activation level, or DOA, of a primitive behavior affects the behavior's contribution to the overall control action by reducing the membership of the action it produces (see Equation 3.21). However, the fact that a DOA only affects the membership of a fuzzy value, and not the fuzzy value itself, leads to problems if one uses the same process to calculate a DOA of a primitive behavior. To illustrate this point, consider the previously discussed situation where B_1 has been given a `LOW` weight and has assigned a `HIGH` weight to the primitive behavior β_2. The `LOW` DOA of the composite behavior reduces the membership, visually represented as the height, of β_2's `HIGH` DOA (see Figure 3.12a). However, once the DOA is defuzzified to a crisp value, the fact that it's membership was reduced is not evident since it still received a DOA of `HIGH`. What is needed is a mechanism which uses the `LOW` DOA of the composite behavior B_1 to reduce the actual defuzzified value of

β_2's DOA (see Figure 3.12b).

Any modifications made to the current process to produce the desired result must ensure the following conditions hold:

1. The current process produces correct results for the simple case of a two-level hierarchy consisting of a single composite behavior coordinating only primitive behaviors. Any modifications made must change only the output values of a DOA calculation and not the output values of a primitive behavior. Since the output values of primitive behaviors are actual motor control actions, these output values should not be altered.

2. In general, more than one composite behavior could potentially coordinate a single primitive behavior. Any modifications made must account for the contribution of DOA values from composite behaviors with different DOA's themselves.

The first condition can be met by simply only modifying output values not used for motor control. In practice, a modification could be applied only when an output value is used as a DOA since the defuzzification process is generic and used for both DOA's and motor control. The second condition precludes any solution that only considers the simple case where a primitive is coordinated by a single composite behavior. In this simple case, the composite behavior's DOA could simply be used to scale a primitive behavior's DOA, as in the following equation:

$$\alpha'_p = \alpha_p \cdot \alpha_c \tag{3.23}$$

where α_p is the primitive behavior's calculated DOA, α_c is the composite behavior's DOA, and α'_p is the final DOA of the primitive behavior. However, since, in general, more than one composite behavior could be coordinating a given primitive behavior, the DOA of each parent composite behavior must be considered in conjunction with the weight it assigns to the primitive behavior.

To achieve the desired result, while ensuring that the previous conditions hold, adaptive fuzzy behavior hierarchies can be extended in the following manner. First, the same process

for calculating the output of a primitive behavior can be used to calculate a behavior's DOA. For example, the DOA given to a primitive behavior by a composite behavior can be calculated using the following:

$$\alpha_p = \frac{\int_{u \in U} u \sum_{c \in C} \alpha_c \cdot \mu_{\tilde{B}_c}(u)}{\int_{u \in U} \sum_{c \in C} \alpha_c \cdot \mu_{\tilde{B}_c}(u)} \tag{3.24}$$

where $\mu_{\tilde{B}_c}(u)$ denotes the membership of a given DOA value assigned by composite behavior B_c. Although the above equation considers the calculation of a primitive behavior's DOA, it can also be used to calculate a composite behavior's DOA. This DOA must now be scaled by the DOA's of the parent composite behaviors. To calculate the scaling factor, we can use the following:

$$S_p = \frac{\sum_{c \in C} \alpha_c \int_{u \in U} u \cdot \mu_{\tilde{B}_c}(u)}{\sum_{c \in C} \int_{u \in U} u \cdot \mu_{\tilde{B}_c}(u)} \tag{3.25}$$

where S_p is the scaling factor for the primitive behavior p. The final DOA of primitive behavior p is computed using the results from Equation 3.23 and 3.25:

$$\alpha'_p = \alpha_p \cdot S_p \tag{3.26}$$

Note that S_p equals α_c in the simple case of only one composite behavior coordinating a given primitive behavior. In this simple case, we recover the intuitive calculation of Equation 3.23. The addition of the scaling factor S_p allows the DOA of the parent composite behavior to cascade down the hierarchy and affect the DOA of the primitive behavior, resulting the desired outcome illustrated in Figure 3.11b.

3.4 Creating Agents Using Adaptive Fuzzy Behavior Hierarchies

In order to provide primitive and composite behaviors with varying levels of state information gathered from the sensors, a sensor hierarchy mirroring the behavior hierarchy was created. This not only ensured that behaviors received the state information relevant only to their current primitive or composite task, but it also allowed for the abstraction of the state information (see Section 6.2). The architecture of an agent using a generic, adaptive fuzzy behavior hierarchy is depicted in Figure 3.13. The order of execution of an agent is as follows:

1. The fuzzy input and output values are reset from the previous execution to prevent past states and actions from interfering with the current timestep.

2. State information is gathered from the sensors and processed for use in the primitive and composite behaviors. Sensors are organized such that those providing state information to primitive behaviors receive low-level data, while those that provide state information to composite behaviors receive data that has already been processed by previous sensors.

3. The behavior hierarchy is evaluated in a top-down fashion. The composite behavior at the highest level of the hierarchy is executed first and then execution continues with behaviors in each lower level until the primitive behaviors are executed. There is no guaranteed order of execution of behaviors within the same hierarchy level. A behavior is not executed if its DOA is below a specified threshold or it is not explicitly given a weighting.

4. The fuzzy output of the primitive behaviors is defuzzified to determine the motor commands of the agent's action.

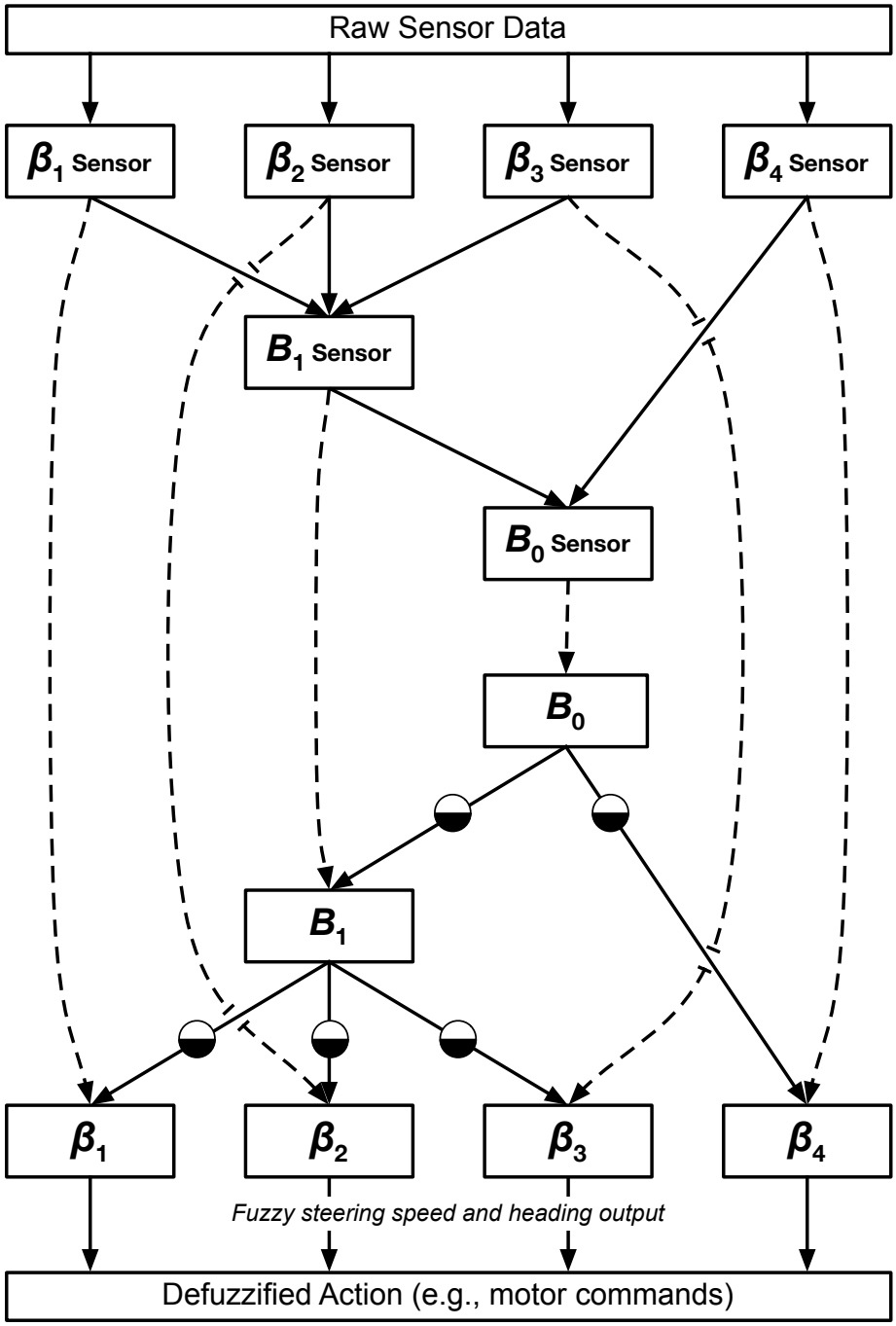


Figure 3.13: The above diagram depicts the architecture of an agent using a generic, adaptive fuzzy behavior hierarchy. Lines from low-level sensors to high-level sensors denote the progressive abstraction of sensor data. Dashed lines from sensors to behaviors denote the use of sensor (or state) information by a behavior. Lines from high-level behaviors to low-level behaviors denote behavior modulation. Note that a sensor hierarchy mirrors the behavior hierarchy and is responsible for providing processed state information to the appropriate primitive and composite behaviors. The sensor hierarchy is important when various abstractions of the state information are used (see Section 6.2).

CHAPTER 4

Navigation Problem Domains

For this work, a number of autonomous agent navigation problem domains were used. In each domain, an agent was given a complex task that was composed of N primitive tasks, where each primitive task was an Markov decision process, or MDP, and, in general, CINE in nature. An MDP M is defined as a discrete-time process with a set of states S, a set of actions A, a transition function $T(s,a,s')$, and a set of rewards R. The set of states and the set of actions are referred to as the state and action spaces, respectively. The transition function describes the probability that a given action a, taken in state s, will lead to state s' at the next timestep. The reward $R(s,a,s')$ is the reward received after transitioning from state s to state s' by taking action a. While not indicative of autonomous agent navigation problems in general, each MDP used a fully observable state. This was done to ensure that any performance difference between controllers for the same set of primitive tasks was only due to the state and action abstractions used.

We considered the case where the N primitive tasks can be, and frequently are, active concurrently, interfere with one another, and have no termination criteria and are considered non-episodic (or CINE). The only exception was the GOALSEEK task which terminated when an agent reached the goal location. The state space S for each primitive task used was distinct and local to that specific task, that is, $S_x \cap S_y = \emptyset$ for any primitive tasks $x, y \in N$. This property is not necessary in the general case, but is indicative of the state spaces used in the work presented here. While the action space for each primitive task may vary between primitive actions, we considered the more difficult problem domains where the same action space was shared among all primitive tasks, that is, $A_x = A_y$ for any primitive tasks $x, y \in$

N. For example, if both a COLLISIONAVOIDANCE and GOALSEEK primitive task were active, only the state space for the COLLISIONAVOIDANCE task contained information regarding potential collisions while only GOALSEEK's state space contained information regarding the goal location. Both primitive tasks would share the same action space, which is composed of steering motor control actions.

The composition of these N primitive tasks forms a *composite task* which attempts to take an action at each timestep that maximizes the summed expected reward of each primitive task:

$$R(s,a,s') = \sum_{i}^{N} R_i(s_i,a,s_i'). \tag{4.1}$$

An important aspect of this combination of primitive tasks is that an action that maximizes the reward for one primitive task could result in a penalty for another primitive task and, therefore, cause interference between the primitive tasks. While each primitive task had a (relatively) small state space, the state space for the composite task was the cross product of the state space for each primitive task: $S = S_1 \times S_2 \times \ldots \times S_N$. When this combined state space, referred to as the *joint state space*, was combined with the low-level action space, the resulting complexity can make the traditional development of an effective controller impractical. In order to manipulate the complexity of each task without fundamentally altering the task itself, both two and three-dimensional environments were used. The addition of the third dimension increased not only the state and action spaces, but also allowed for more complex interactions between the agent(s) and the environment, especially when using machine learning techniques to develop controllers.

Two sets of problem domains were used in this work. In the first, a single agent was given composite tasks in environments without any other agents present with which to interact. These tasks represented the least complex problem domains and provided examples in which traditional development methods are potentially practical. In the second set, a team of agents was given composite tasks that required successful coordination between the agents. These tasks represented more complex problem domains and provided exam-

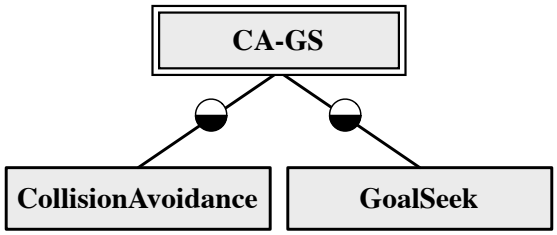


Figure 4.1: In the CA-GS behavior hierarchy, the CA-GS composite behavior weights the COLLISIONAVOIDANCE and GOALSEEK primitive behaviors. Half-filled circles denote the weights used by behavior modulation.

ples in which traditional development methods are impractical. This complexity extended beyond simply the size of the state-action space since a successful controller must deal with agent-to-agent interactions. Note, however, that the state space for the multi-agent tasks is not the joint state space of all the agents. The state variables used by the controllers are independent of the number of other agents in the environment as they represent the aggregate information of the team.

4.1 Single Agent Problem Domains

In the first single-agent composite task, an agent navigated towards a goal location while avoiding any obstacles in its path. This composite task, denoted CA-GS, was the combination of the COLLISIONAVOIDANCE and GOALSEEK primitive tasks. The state information local to the COLLISIONAVOIDANCE task consisted of the relative direction to the closest potential collision and the estimated time until collision. The state information local to the GOALSEEK task consisted of the relative direction to the goal location and the estimated time of arrival at the goal location given only the current speed of the agent. All state information was normalized and measured relative to the agent's position and orientation. This was done to decouple the learned behavior from the specifics of the environment. A more detailed description of the state space for these primitive tasks, and the others discussed below, can be found in Section 4.3.

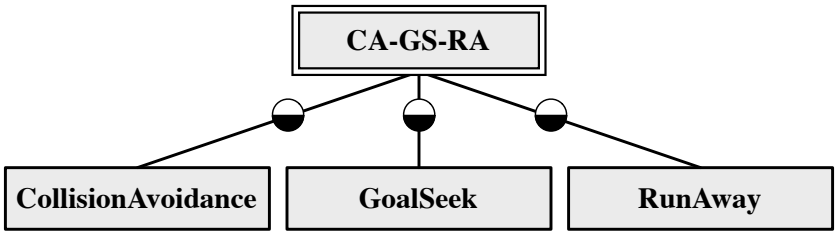


Figure 4.2: The CA-GS-RA behavior hierarchy adds the RUNAWAY primitive behavior to the existing CA-GS behavior hierarchy. Half-filled circles denote the weights used by behavior modulation.

In the fuzzy behavior hierarchy for the CA-GS composite task, primitive behaviors were created at the lowest level for the COLLISIONAVOIDANCE and GOALSEEK primitive tasks (see Figure 4.1). A composite behavior for the CA-GS composite task was then created at the level above the primitive behaviors and was responsible for weighting the primitive behaviors properly, given the current state of the agent with respect to the composite task as a whole.

In the second single-agent composite task, a third primitive task was added to the previous two. In this new primitive task, denoted RUNAWAY, the agent must avoid approaching too close to "hazardous" objects in the environment. The hazardous objects were not physical objects like obstacles with which the agent could collide, but instead represented areas that could be dangerous to the agent like areas of high-traffic or with difficult terrain. The state information local to the RUNAWAY task consisted of the relative direction and magnitude of a repulsive "force" which effectively acted to steer the agent away from all the sensed hazardous objects. The new composite task was denoted CA-GS-RA. The fuzzy behavior hierarchy for the CA-GS-RA composite task was similar to that of the CA-GS hierarchy with the addition of the RUNAWAY primitive behavior (see Figure 4.2).

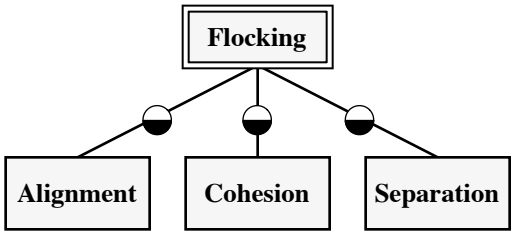
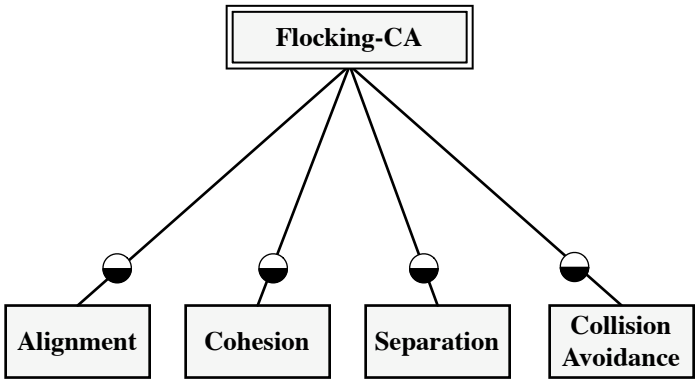


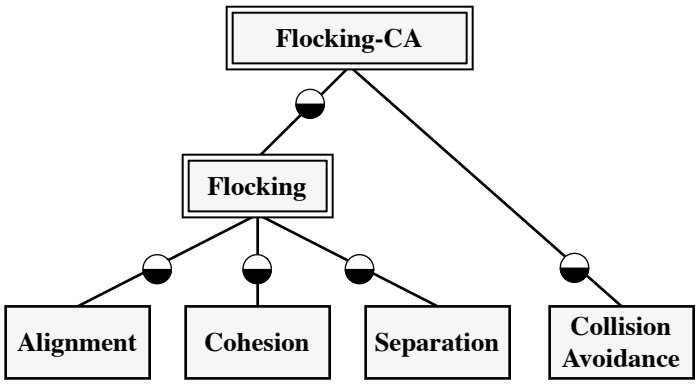
Figure 4.3: The FLOCKING composite behavior composes the ALIGNMENT, COHESION, and SEPARATION primitive behaviors by using weights, denoted as half-filled circles.

4.2 Multi-Agent Problem Domains

In the first multi-agent composite task, a team of homogeneous agents must move together as a single unit, or *flock*, without explicit communication. This composite task, denoted FLOCKING, approximated the movement of flocks of birds or schools of fish [70, 71] and was a combination of the ALIGNMENT, COHESION, and SEPARATION primitive tasks. In the ALIGNMENT primitive task, the agents were given the task of steering in the same direction and at the same speed as the rest of the team. The state information local to this task consisted of the normalized relative differences in speed and heading. In the COHESION primitive task, the agents were given the task of steering towards the other agents in the team in an effort to remain close to the team. The state information local to the COHESION task consisted of the normalized distance and relative heading to the mean position of the other sensed teammates. Lastly, in the SEPARATION primitive task, the agents were given the task of steering away from other agents on the team which were "too close" in an effort to maintain a safe, minimum separation and prevent crowding. The state information local to the SEPARATION task consisted of the strength and relative direction of a repulsive "force" from the other sensed teammates. Agents relied only on the state information provided by sensors and did not communicate. Note that the goals of the ALIGNMENT and SEPARATION primitive tasks are diametrically opposed. Therefore, for FLOCKING to be successful, a policy that was able to effectively balance the two was necessary.



(a) 2-level hierarchy



(b) 3-level hierarchy

Figure 4.4: The two alternatives for implementing the FLOCKING-CA composite behavior are shown. In the first, the FLOCKING-CA composite behavior composes the ALIGNMENT, COHESION, SEPARATION, and COLLISIONAVOIDANCE primitive behaviors. In the second, FLOCKING-CA composes the FLOCKING composite behavior with the COLLISIONAVOIDANCE primitive behavior.

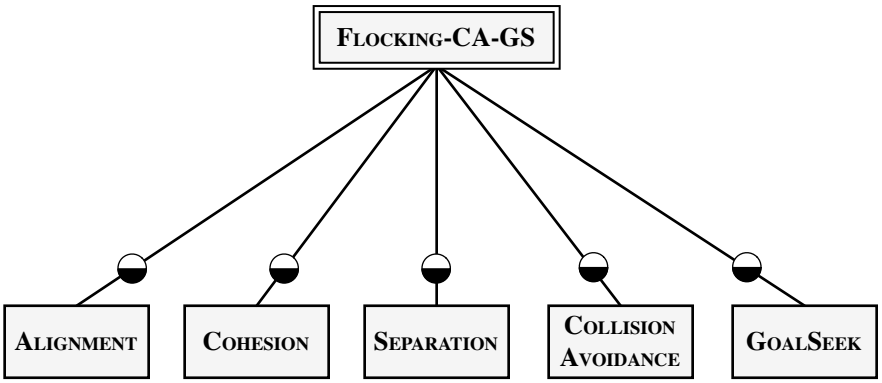
In the fuzzy behavior hierarchy for the FLOCKING composite task, primitive behaviors were created at the lowest level for the ALIGNMENT, COHESION, and SEPARATION primitive tasks (see Figure 4.3). A composite behavior for the FLOCKING composite task was then created at the level above the primitive behaviors and was responsible for weighting ALIGNMENT, COHESION, and SEPARATION properly, given the current state of the composite task.

In the next multi-agent composite task, we added the primitive task of COLLISIONAVOIDANCE to the FLOCKING composite task. In this task, each agent was tasked with

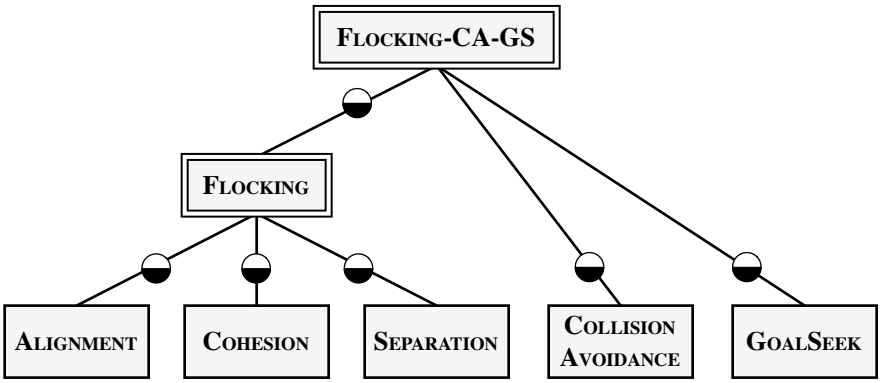
avoiding collisions with other agents and with obstacles in the environment in addition to performing FLOCKING. The COLLISIONAVOIDANCE primitive task used the local state information consisting of the direction and estimated time until the nearest collision. In fact, the task did not differentiate between collisions with other agents or obstacles. This new composite task, denoted FLOCKING-CA, presented the option of adding a new composite behavior, and, therefore, another level to the hierarchy (see Figure 4.4). Since the task was ignorant of the concept of a team and teammates, an argument can be made that the COLLISIONAVOIDANCE primitive task should be considered separately from the FLOCKING primitive tasks. The use of an additional composite behavior at a higher level in the hierarchy did not only simplify the action space of the FLOCKING-CA composite task, but it also had the potential simplify the state space if the full joint state space of the composite task was not necessary for effective control. Furthermore, this hierarchical decomposition enabled existing policies for the FLOCKING task to be reused for the FLOCKING-CA task.

To further increase the complexity, the GOALSEEK primitive task was added to the previous composite task to create the FLOCKING-CA-GS composite task. While the COLLISIONAVOIDANCE primitive task could have actually assisted the task of FLOCKING by providing another means of avoiding collisions between members of the team in addition to the SEPARATION task, the addition of the GOALSEEK complicated the FLOCKING task. Since the state space for the GOALSEEK primitive task only included the relative direction to the goal location and the estimated time of arrival at the goal location, it was ignorant of any other agents in the environment. While the same potential for interference existed in the CA-GS or CA-GS-RA composite tasks, the presence of multiple agents navigating towards the same goal location only served to increase the potential for collision. As with the FLOCKING-CA task, the clear separation between the FLOCKING composite task and the COLLISIONAVOIDANCE and GOALSEEK primitive tasks offered the potential for creating a separate composite behavior for coordinating the respective behaviors.

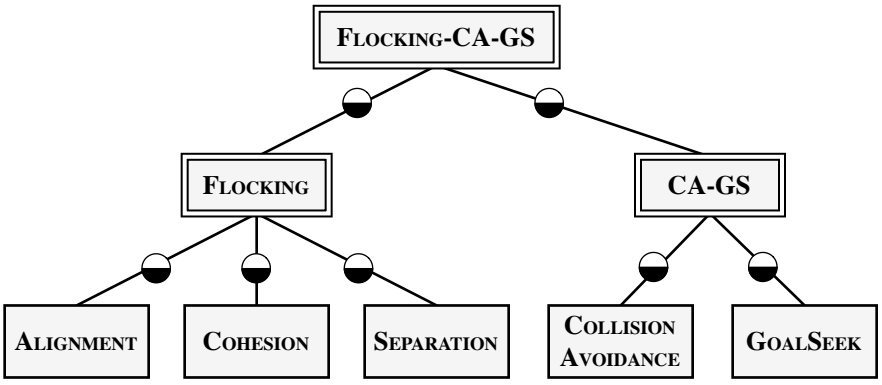
In the last multi-agent composite task, the RUNAWAY primitive task was added to the



(a) 2-level hierarchy

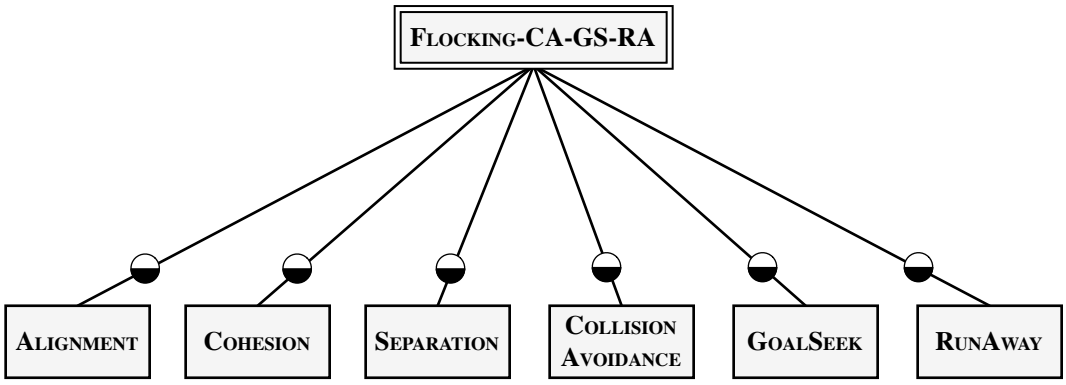


(b) 3-level hierarchy reusing one composite behavior

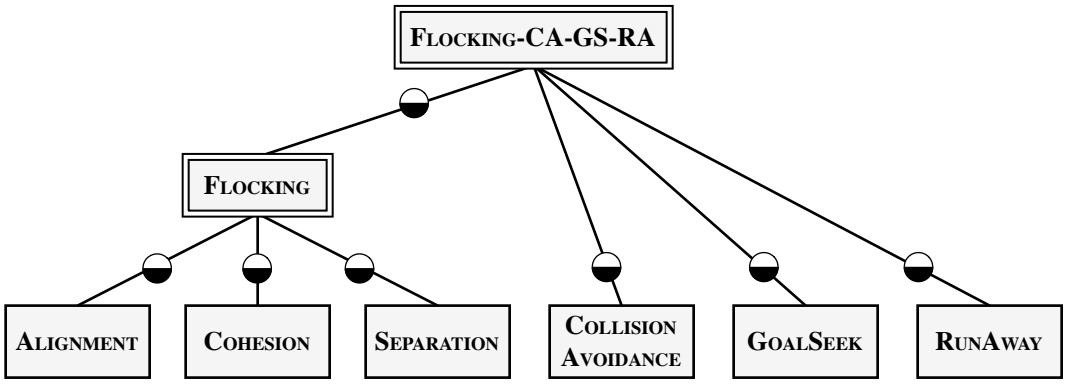


(c) 3-level hierarchy reusing two composite behaviors

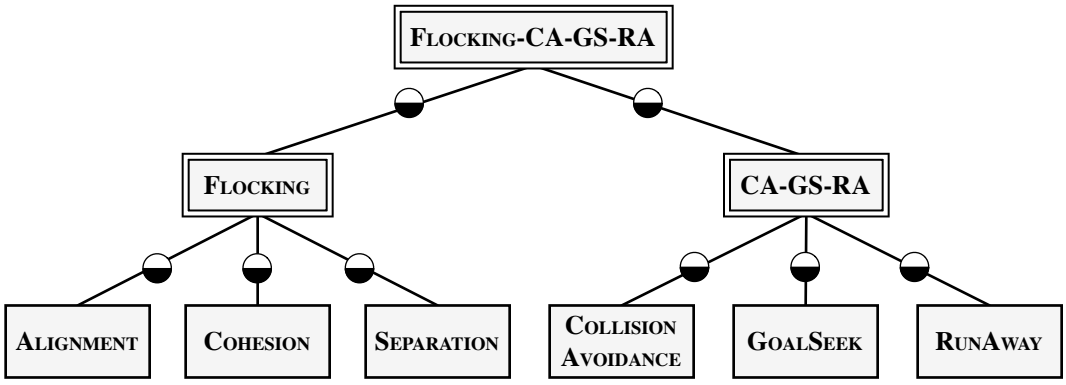
Figure 4.5: Three alternatives for implementing the FLOCKING-CA-GS composite behavior are shown. Each is similar to the corresponding hierarchy for the FLOCKING-CA composite task with the addition of the GOALSEEK primitive behavior



(a) 2-level hierarchy



(b) 3-level hierarchy reusing one composite behavior



(c) 3-level hierarchy reusing two composite behaviors

Figure 4.6: Three alternatives for implementing the FLOCKING-CA-GS-RA composite behavior are shown. Each is similar to the corresponding hierarchy for the FLOCKING-CA-GS composite task with the addition of the RUNAWAY primitive behavior

FLOCKING-CA-GS composite task to create the FLOCKING-CA-GS-RA composite task. As with the previous two composite tasks, a separate composite behavior which coordinated the FLOCKING composite task and the COLLISIONAVOIDANCE, GOALSEEK, and RUNAWAY primitive behaviors was possible. While the same potential benefits exist, the large number of primitive tasks which must be coordinated had the potential exaggerate the results. As is described in Chapter 7, it was at this point that the complexity of the composite task's joint state space became too large for traditional methods of developing effective controllers to be practical and alternative methods were required (see Appendix A for more detailed information). The use of this composite task represented the upper limit of complexity used in this work.

4.3 Primitive Task State Information

The state information for each primitive task was described in egocentric terms to facilitate generalization and to simplify the state space. Furthermore, time estimates for events such as a potential collision or arrival at the goal location were also used to facilitate generalization and to simplify the state space. Otherwise, both a distance and the agent's current speed would be required. Before each piece of state information was used by a controller, the value was ensured to be within an interval with a maximum value and then normalized. This ensured that environment and agent specific information (e.g., maximum sensor range and minimum safe distance) were not directly used by the controller. This promotes generalization in the controller and reuse in other composite tasks and with different agents. However, the fact that state information was ensured to be within an interval with a maximum level contributed situations where different states which require different actions appeared to be the same state. This problem of perceptual aliasing could negatively impact the learning of an effective controller [99]. The motor control actions available to each primitive task were also relative to the agent's current state and specified the change in speed and heading.

Table 4.1: The different state information relevant to each primitive task are shown. In addition, the number of discrete values for each variable is also shown. When fuzzy logic is used, the values are not crisp. See Appendix E for more details on the fuzzy membership functions used. Note that Φ related variables are only present in three-dimensional environments.

Primitive Task	**State Variable**	**Discrete Values**
COLLISIONAVOIDANCE	Time until collision	5
	Collision direction Θ	7
	Collision direction Φ	5
GOALSEEK	Arrival time at goal	5
	Goal direction Θ	7
	Goal direction Φ	5
RUNAWAY	Runaway strength	5
	Runaway direction Θ	7
	Runaway direction Φ	5
ALIGNMENT	Speed difference	5
	Heading difference Θ	7
	Heading difference Φ	5
COHESION	Arrival time at mean team position	5
	Mean team position direction Θ	7
	Mean team position direction Φ	5
SEPARATION	Separation strength	5
	Separation direction Θ	7
	Separation direction Φ	5

The state information relevant to each primitive task is shown in Table 4.1. Note that Θ has seven linguistic values while Φ only has five. This is due to the fact that Θ was defined on larger interval $[-\pi, \pi]$ and needs more resolution than Φ which was only defined on the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$. The state information relevant to each primitive task was the only state information used by the associated primitive behavior. The state information for a composite task was a combination of all the state information for the primitive tasks comprising the composite task. A composite behavior may use all the state information or a subset of the information in an abstracted form (see Chapter 6). Note that state information using Φ directions were only present in three-dimensional environments.

CHAPTER 5

Development of Controllers

While the use of an adaptive fuzzy behavior hierarchy does simplify the design of a controller, there still exists the challenge of developing an *effective* controller. The first step in the process is to identify the input and output values, or linguistic variables, of the system. Appropriate linguistic values and membership functions for these linguistic variables must also be defined. This requires a familiarity with not only the current task, but also an understanding of how the membership functions will affect the resulting controller. The next step is to design a set of fuzzy rules to produce the desired behavior[1]. This also requires expert knowledge of the current task, but at a far deeper level than that required in the creation of the membership functions. Development of the fuzzy rules for a behavior, especially manual development, requires an understanding of all the competing aspects of a task. Rarely, if ever, are either the membership functions or the fuzzy rules correct in their initial form. An iterative "test-debug-tune" cycle is usually required before a controller is effective [93]. The amount of time and effort required by this development cycle is dependent on the complexity of the task and the amount of expert knowledge in the problem domain that is available.

The development of primitive behaviors is straightforward since the primitive tasks they accomplish are, by definition, simple. The simplicity of the task itself means that manually creating fuzzy rules using a heuristic is entirely practical. This is especially true if one believes that alternative approaches, such as reinforcement learning, should only be used if a solution is not already known. Similarly, the simplicity of the task means

[1]While the organization of the behavior hierarchy itself must precede this step, we assume that such a process has already been done in the process of creating a composite task by composing already defined primitive tasks.

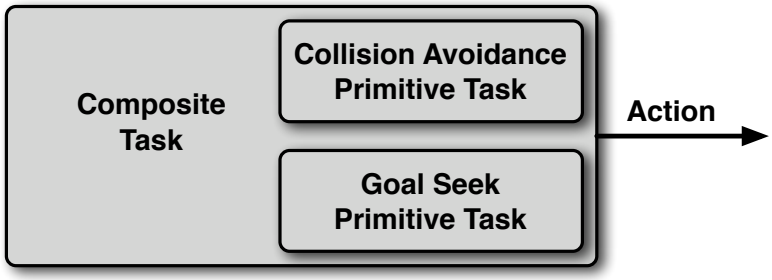
that the linguistic values and membership functions are relatively easy to create. While fine tuning of the rules and membership functions may be required, the effort required is usually minimal. In his original work on adaptive fuzzy behavior hierarchies, Tunstel manually developed primitive behavior rulesets for use on an actual robot with great success.

The development of composite behaviors, however, is far more complex than the development of primitive behaviors. While the overall composite task may be simple conceptually, its implementation can be quite complex. As primitive tasks are added to the composite task, the availability of expert knowledge decreases significantly. For example, although manually developing an effective composite behavior for the CA-GS composite task is more difficult than developing controllers for the individual primitive tasks, it is possible since only two primitive behaviors require coordination. On the other hand, manually developing an effective composite behavior for the FLOCKING-CA-GS composite task is impractical at best. As a result, for the development of composite behaviors to be practical, alternatives to their manual development must be used.

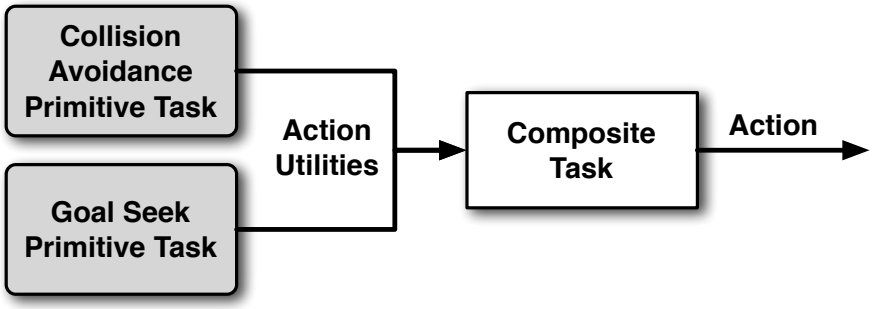
There are two aspects of developing a behavior, whether it be composite or primitive, that one must consider. The first is the fuzzy ruleset for the behavior and the second is the linguistic values and membership functions used by the fuzzy rules. Although there is work in the field that seeks to develop both either in parallel [21] or sequentially in an iterative process (see Section 2.4), this work used manually developed linguistic values and membership functions and focused on automatically creating fuzzy rulesets. This was done to minimize the number of sources which could potentially affect the performance of various approaches in different ways. Two different methods of automatically creating composite behaviors were used: reinforcement learning and grammatical evolution.

5.1 Composite Reinforcement Learning

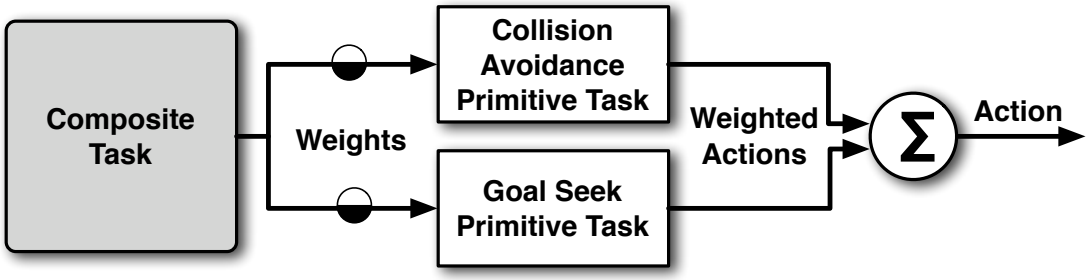
While reinforcement learning is capable, in general, of learning effective control policies, many of the more popular reinforcement learning approaches are either not applicable to



(a) Monolithic Reinforcement Learning



(b) Modular Reinforcement Learning



(c) Composite Reinforcement Learning

Figure 5.1: A comparison of the different reinforcement learning algorithms discussed in this work is shown using a simple COLLISIONAVOIDANCE-GOALSEEK composite task. The shaded tasks are where policy learning occurs in each algorithm. The half-filled circles denote the weights used to compose actions from primitive task policies for composite reinforcement learning.

CINE tasks or are incapable of dealing with the resulting complexity (see Section 2.3). One of the few reinforcement learning approaches that is applicable to the CINE tasks under study here, modular reinforcement learning, has many properties that can prove to be liabilities (see Section 2.3.2). An alternative method is, therefore, needed if reinforcement learning can be used to learn effective control policies for composite tasks built using CINE primitive tasks.

In light of this need, we propose a new reinforcement learning approach that we term *composite reinforcement learning* (see Figure 5.1). Composite reinforcement learning leverages the architecture of the adaptive fuzzy behavior hierarchy to significantly improve the rate at which effective control policies are learned (see Section 3.2). Unlike modular reinforcement learning, composite reinforcement learning does not attempt to learn policies for the primitive tasks simultaneously. Instead, composite reinforcement learning learns an effective control policy for a given composite behavior only and reuses existing implementations of lower-level behaviors. These reused lower-level behaviors are viewed as black boxes and are modulated by the policy being learned. Therefore, instead of learning low-level motor control actions, composite reinforcement learning learns high-level modulation (i.e., weighting) actions on the lower-level behaviors. The reinforcement learning algorithm itself is largely unmodified except that the concept of an action has changed. The policy's actions are now weighting actions and after the policy's action has been taken, the lower-level behaviors are executed and the overall action of the agent is computed. The composite task policy being learned then determines the total reward and updates the relevant Q-value. As a result of this change in definition, the Sarsa update rule can now be described by the following:

$$Q_{ct}(s_{ct}, a_{ct}) \leftarrow (1-\alpha)\, Q_{ct}(s_{ct}, a_{ct}) + \alpha(r + \gamma Q_{ct}(s'_{ct}, a'_{ct})) \tag{5.1}$$

where s_{ct} is the state of the agent with respect to the composite task, a_{ct} is the action

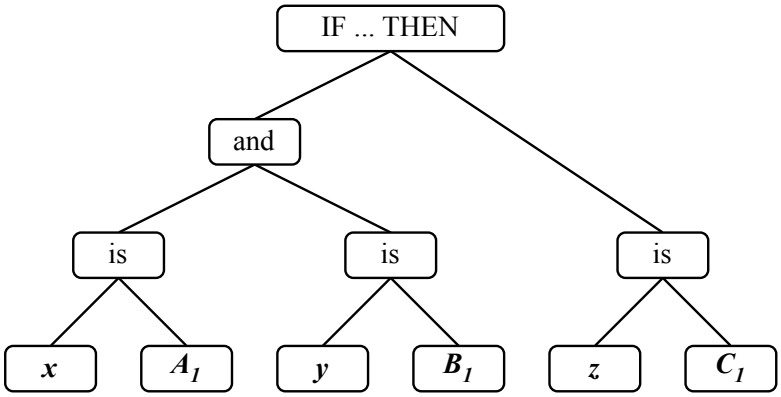
taken for the current composite task which is consists of the modulation of the lower-level behaviors, and $Q_{ct}(s_{ct}, a_{ct})$ is the Q-value of the state and action with respect to the composite task.

Note that the Q-values used by the learned policy are associated only with the modulation actions and not with the actions taken by the lower-level behaviors. While this means that the maximum performance of the learned policy is dependent on the performance of the lower-level behaviors in their associated tasks, in practice this appears to not present problems (see Chapter 7) and offers many benefits over other approaches, such as modular reinforcement learning, especially in the work presented in this dissertation.

One of the most significant benefits is the abstraction of the action space into high-level, "meta-actions." As a result, the reinforcement learner is not required to learn the entire composite task from scratch. Rather, it only needs to learn how to best coordinate lower-level behaviors to accomplish the composite task. In a related benefit, existing behaviors can potentially be reused *without modification* by the learned policy and without specific requirements on their implementation method. Composite reinforcement learning does not require reused behaviors provide any information to aid in the learning or control process (e.g., Q or utility values). As a result, individual behaviors can be developed in isolation, simplifying the development process, using the method most appropriate for the task.

Furthermore, since the action space has been abstracted away from low-level motor control actions, it may be possible to aggressively abstract the agent's state for use in the composite behavior without the corresponding performance penalties commonly associated with perceptual aliasing [99]. This is especially significant in light of our interest in directly comparing the effects of state and action abstraction on a controller's performance and learning rate. Note that this abstraction of the state only occurs in the composite behaviors; primitive behaviors still access the unabstracted state associated with the relevant primitive task to produce control actions.

While the idea of abstracting the action space into meta-actions is not novel and many



IF x is A_1 and y is B_1 THEN z is C_1

Figure 5.2: A sample genetic programming solution representing a fuzzy rule using a tree-like structure of symbols is shown.

hierarchical reinforcement learning approaches use it extensively [18, 42, 72], our formulation of an action is novel. Since most approaches focus on episodic and non-interfering tasks, meta-actions in these approaches represent temporally extended sequences of actions. When a meta-action is executed, the meta-action assumes control of the agent either for the entire sequence of actions or until an event causes the high-level policy to re-examine the agent's state. In contrast, the meta-actions used by composite reinforcement learning are taken every timestep and represent the coordination of lower-level behaviors for that timestep only. In general, no one behavior is given complete control of the agent's actions.

5.2 Grammatical Evolution

Originally, composite behaviors for adaptive fuzzy behavior hierarchies were automatically developed using genetic programming. Genetic programming was used offline to evolve the fuzzy coordination rules for a composite behavior. Evolved rulesets were then evaluated in simulated environments to determine their fitness. Genetic programming uses a tree-like structure of symbols to represent individuals, instead of the bit-string representation commonly used by genetic algorithms [43]. This tree-like structure of symbols is conceptually

similar to the tree-like structure of a fuzzy ruleset. The combination of this tree-like structure and genetic programming's use of symbols instead of numeric values lends itself to the production of fuzzy rulesets (see Figure 5.2). Each branch from the root node in the tree represents a separate rule in the ruleset. The linguistic variables and values used in antecedents and consequents form a terminal set. The rules, antecedents, consequents, and other fuzzy rule components form a function, or non-terminal, set.

While genetic programming was capable of generating effective rulesets in Tunstel's work [93], its use can present difficulties. First, invalid rules can be created if measures aren't taken to ensure that the crossover and mutation operators always produce valid rules. Strongly-typed genetic programming [55] can be used to prevent the creation of invalid rules, but its use necessitates additional computational effort to search for valid crossover points between two individuals. Second, large portions of individuals created by genetic programming are not ever used during evaluation and are referred to as *introns*. These introns serve to protect useful parts of the individual from destructive crossover or mutation. However, introns don't affect the fitness of the individual and can result in wasted computational resources. Furthermore, for an entire population of individuals with a large tree structure, the memory and computational resources necessary can become prohibitive. Since the problem domains under investigation in this work are complex and already require significant computational resources, an alternative approach was desirable.

Since the introduction of adaptive fuzzy behavior hierarchies, a new evolutionary approach has been introduced which has the expressive power of genetic programming with the implementation simplicity of genetic algorithms. Grammatical evolution uses a context-free grammar to map a variable-length bit-string genotype, like those used in genetic algorithms, to a more complex, tree-like structure of symbols, like those produced by genetic programming [58, 79]. Grammatical evolution offers all the benefits of using genetic programming to evolve fuzzy rulesets, without the potential problems. By carefully designing an appropriate grammar, we can ensure that all the rulesets created by grammatical evo-

lution are valid. Furthermore, since an individual is maintained as a simple bit-string, the memory requirements are minimal. Also, as will be shown, introns are only present in the genotype and are easily pruned. In previous work, we demonstrated that grammatical evolution is capable of providing effective fuzzy rulesets in a single-agent navigation problem domain [24].

In grammatical evolution, a context-free grammar is used to map the bit-string genotype to a symbolic phenotype. The Backus Naur Form notation is used to describe the grammars used in this work. Formally, a grammar G is defined by $G = \{N, T, P, S\}$, where N is the set of non-terminals, T is the set of terminals, P is a set of production rules, and S is the start symbol where $S \in N$. For example, consider a simple grammar used to produce fuzzy rules for the GOALSEEK primitive task (see Figure 5.3). The set of non-terminals includes symbols necessary to organize the fuzzy rules such as: *rule*, *antecedent*, and *consequent*. The terminal set includes the linguistic variables and values used in fuzzy rules such as: *goal-direction*, `LEFT`, *steer-speed*, and `FASTER`. In this grammar, the start symbol is a *rule* which is capable of producing other rules. The potential production rules for this grammar are shown in Figure 5.3.

To produce an individual's symbolic phenotype, the grammar's start symbol is replaced using one of the specified production rules. Replacement continues until either there are no more non-terminals to replace or a maximum number of rules is reached. In this process, the genotype is used to select the replacement of a production rule to perform. This is accomplished by organizing the bit-string into codons of 8-bits each. The integer value of the codon is then used to determine which replacement is made. This process continues with the next codon until replacement stops. If the end of the bit-string is reached while non-terminals remain, replacement continues at the beginning of the bit-string. A portion of this process is depicted in Figure 5.4.

Since the genotype is a simple, variable-length bit-string, standard genetic algorithm operators can be used in grammatical evolution. No modifications are necessary to en-

T = {LEFT, CENTER, RIGHT, NOW, SOON, DISTANT, SLOWER, SAME, FASTER}

N = {⟨*rule*⟩, ⟨*antecedent*⟩, ⟨*consequent*⟩, ⟨*goal-dir*⟩, ⟨*goal-arrival-time*⟩, ⟨*steer-speed*⟩, ⟨*steer-dir*⟩}

S = ⟨*rule*⟩

P is represented as:

⟨*rule*⟩ ::= ⟨*antecedent*⟩ ⟨*consequent*⟩
| ⟨*antecedent*⟩ ⟨*consequent*⟩; ⟨*rule*⟩

⟨*antecedent*⟩ ::= ⟨*antecedent*⟩ ⟨*antecedent*⟩
| ⟨*goal-dir*⟩
| ⟨*goal-arrival-time*⟩

⟨*consequent*⟩ ::= ⟨*consequent*⟩ ⟨*consequent*⟩
| ⟨*steer-speed*⟩
| ⟨*steer-dir*⟩

⟨*goal-dir*⟩ ::= goal-dir(LEFT)
| goal-dir(CENTER)
| goal-dir(RIGHT)

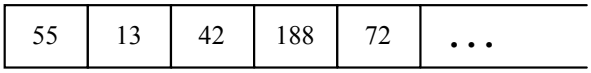
⟨*goal-arrival-time*⟩ ::= goal-arrival-time(NOW)
| goal-arrival-time(SOON)
| goal-arrival-time(DISTANT)

⟨*steer-speed*⟩ ::=steer-speed(SLOWER)
| steer-speed(SAME)
| steer-speed(FASTER)

⟨*steer-dir*⟩ ::= steer-dir(LEFT)
| steer-dir(CENTER)
| steer-dir(RIGHT)

Figure 5.3: A sample grammatical evolution grammar which could be used in generating fuzzy rules for the GOALSEEK primitive task is shown.

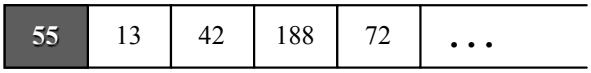
1. Replacement is initialized by converting the bit-string to a codon-string



   and the grammar's start symbol:

   $\langle rule \rangle$

2. Replacement starts by using the first codon to select a replacement from the start symbol's production rule:



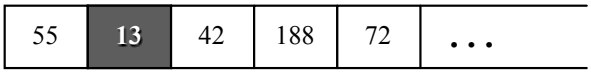
   The index of the replacement is chosen by calculating the codon's value modulus the number of replacement options:

$$55 \bmod 2 = 1$$

   which then replaces the current non-terminal:

   IF $\langle antecedent \rangle$ THEN $\langle consequent \rangle$; $\langle rule \rangle$

3. Replacement continues by using the second codon to select a replacement for the $\langle antecedent \rangle$ non-terminal:



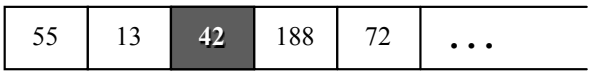
$$13 \bmod 3 = 1$$

   which then replaces the non-terminal:

   IF $\langle goal\text{-}dir \rangle$ THEN $\langle consequent \rangle$; $\langle rule \rangle$

Figure 5.4: A sample grammatical evolution replacement process using the grammar defined in Figure 5.3 is shown.

4. The third codon is then used to select a replacement for the ⟨*goal-dir*⟩ non-terminal:



$$42 \bmod 3 = 0$$

which then replaces the non-terminal:

IF goal-dir(LEFT) THEN ⟨*consequent*⟩; ⟨*rule*⟩

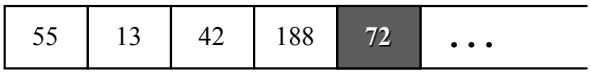
5. Since the last replacement produced a terminal, replacement moves on to the ⟨*consequent*⟩ non-terminal and the fourth codon is then used to select a replacement:

$$188 \bmod 3 = 2$$

which then replaces the non-terminal:

IF goal-dir(LEFT) THEN ⟨*steer-dir*⟩; ⟨*rule*⟩

6. The fifth codon is then used to select a replacement for the ⟨*steer-dir*⟩ non-terminal:



$$72 \bmod 3 = 0$$

which then replaces the non-terminal:

IF goal-dir(LEFT) THEN steer-dir(LEFT); ⟨*rule*⟩

7. Replacement continues . . .

Figure 5.4: A sample grammatical evolution replacement process using the grammar defined in Figure 5.3 is shown *(continued)*.

sure that valid rules are produced because the genotype does not represent a ruleset. By carefully designing the grammar used, one can ensure that the ruleset generated by the grammar is valid. As a result, the standard one-point crossover operator is frequently used in grammatical evolution. Mutation in grammatical evolution is the same as that found in genetic algorithms and randomly flips bits in the bit-string. After a bit-string has been used to produce a phenotype, the unused codons in the genotype can be pruned as they represent introns. This is much simpler than a similar process for genetic programming would be.

Figure 5.5 shows the production rules used to generate fuzzy rulesets for the two-dimensional CA-GS composite behavior. Note that there is a duplicate choice in the production rule for the grammar's start symbol $\langle rule \rangle$. The addition of a duplicate choice was made to bias replacement towards the generation of multiple rules. Exploratory experiments demonstrated that without this bias, the generated rulesets had very few rules and sub-optimal fitness. While such a bias is not generally needed in a grammar, the fact that we are randomly generating rules using the grammar necessitates the bias. An example would be to give higher weight to the letter 'e' when randomly generating words for the English language since it is the most common letter. Since grammatical evolution does not provide a means of giving weight to specific choices for replacement, this bias was manually implemented by adding the additional replacement choice. This addition did not fundamentally alter the grammar, it simply increased the probability of choosing a replacement which resulted in additional rules. There has been recent work on adding attributes to grammars which may address this need [12, 13], but such an investigation is beyond the scope of this work.

⟨*rule*⟩ ::= ⟨*antecedent*⟩ ⟨*consequent*⟩ ⟨*rule*⟩
| ⟨*antecedent*⟩ ⟨*consequent*⟩ ⟨*rule*⟩
| ⟨*antecedent*⟩ ⟨*consequent*⟩

⟨*antecedent*⟩ ::= ⟨*antecedent*⟩ ⟨*antecedent*⟩
| VERY (⟨*antecedent*⟩)
| NOT (⟨*antecedent*⟩)
| ⟨*collision-dir*⟩
| ⟨*time-till-collision*⟩
| ⟨*goal-dir*⟩
| ⟨*goal-arrival-time*⟩

⟨*consequent*⟩ ::= ⟨*consequent*⟩ ⟨*consequent*⟩
| VERY (⟨*consequent*⟩)
| ⟨*relative-steer-speed*⟩
| ⟨*steer-dir*⟩

⟨*collision-dir*⟩ ::= collision-dir(BACK_LEFT)
| collision-dir(LEFT)
| collision-dir(SMALL_LEFT)
| collision-dir(CENTER)
| collision-dir(SMALL_RIGHT)
| collision-dir(RIGHT)
| collision-dir(BACK_RIGHT)

⟨*time-till-collision*⟩ ::= time-till-collision(NOW)
| time-till-collision(REAL_SOON)
| time-till-collision(SOON)
| time-till-collision(LONG_TIME)
| time-till-collision(DISTANT)

⟨*goal-dir*⟩ ::= goal-dir(BACK_LEFT)
| goal-dir(LEFT)
| goal-dir(SMALL_LEFT)
| goal-dir(CENTER)
| goal-dir(SMALL_RIGHT)
| goal-dir(RIGHT)
| goal-dir(BACK_RIGHT)

Figure 5.5: The production rules used for evolving a monolithic fuzzy ruleset for the two-dimensional CA-GS composite behavior is shown. The grammar's start symbol is ⟨*rule*⟩.

⟨*goal-arrival-time*⟩ ::= goal-arrival-time(NOW)
| goal-arrival-time(REAL_SOON)
| goal-arrival-time(SOON)
| goal-arrival-time(LONG_TIME)
| goal-arrival-time(DISTANT)

⟨*relative-steer-speed*⟩ ::= steer-speed(MUCH_SLOWER)
| steer-speed(SLOWER)
| steer-speed(SAME)
| steer-speed(FASTER)
| steer-speed(MUCH_FASTER)

⟨*steer-dir*⟩ ::= steer-dir(LEFT)
| steer-dir(SMALL_LEFT)
| steer-dir(CENTER)
| steer-dir(SMALL_RIGHT)
| steer-dir(RIGHT)

Figure 5.5: The production rules used for evolving a monolithic fuzzy ruleset for the two-dimensional CA-GS composite behavior is shown. The grammar's start symbol is ⟨*rule*⟩. *(continued)*

CHAPTER 6

Implementation and Evaluation

To evaluate the effects of state and action abstraction on the process of developing controllers for composite tasks, a series of experiments were performed in which controllers where automatically developed for each primitive and composite task using reinforcement learning and grammatical evolution (see Chapter 4). First, controllers for each of the primitive tasks were developed. The results of these experiments provided a baseline measure for the effort required to create an effective controller for a primitive task in isolation. Next, controllers for each of the composite tasks were developed.

Experimental runs were evaluated using two different metrics. The first metric used was the best generalized performance of the agent controllers. This generalized performance was determined by executing agent controllers in environments which were different than the ones used in their development. For controllers developed using reinforcement learning, the controller's performance was the mean undiscounted, total reward of the agent in all the environments. For controllers developed using grammatical evolution, the controller's performance was the mean fitness of the agent in all the environments. The second metric used was the computational effort used to develop the controllers. For controllers developed using reinforcement learning, the number of updates to the Q-values was used to measure the rate of development, while the number of generations was used for controllers developed using grammatical evolution.

While the number of training episodes could have been used as a basis for comparing the rate of development for controllers developed using reinforcement learning, it would not have been a fair comparison since training episodes did not contain equal numbers

of learning experiences. For example, the number of learning experiences for an episode with an early collision was far less than for an episode with a late collision. Since the fitness of individual solutions in grammatical evolution (in this case, a set of fuzzy rules) was determined by the overall performance in an episode, the performance per timesteps evaluated was not applicable (which would be the closest measure to reinforcement learning's updates). Note that as a consequence, reinforcement learning results and grammatical evolution results cannot be directly compared.

6.1 Evaluation Environments

A lightweight, custom simulator was designed and implemented for the development and evaluation of agent controllers. It was designed with a focus on speed and configurability in an effort to maximize the variety of experiments with which it could be used. As a result, it provided a low-fidelity simulation environment in which kinematics were simulated, but dynamics were not. The simulator supported both two and three-dimensional continuous environments which were defined by simple XML files. All sensors in the environment were idealized and, therefore, had no noise in the sensor readings. Agents were implemented using configurable modules which could specify sensors, behaviors, or other routines necessary for operating in the simulator. Since each module was configurable, replacing heuristic behaviors with reinforcement learning policies or evolved rulesets was simple and did not require the creation of special "learning" agents.

For each composite task, forty environments were randomly generated. Agents were given random positions and orientations within a specified region of the environment. If a goal location was required, it was randomly placed within a specified distance interval from the agent(s). If obstacles were required, a random number of obstacles were generated and given random positions in an area surrounding the agent(s) and goal location. The same procedure was followed for hazardous objects, if required. An example of a randomly generated environment for the CA-GS-RA composite task is shown in Figure 6.1. A listing

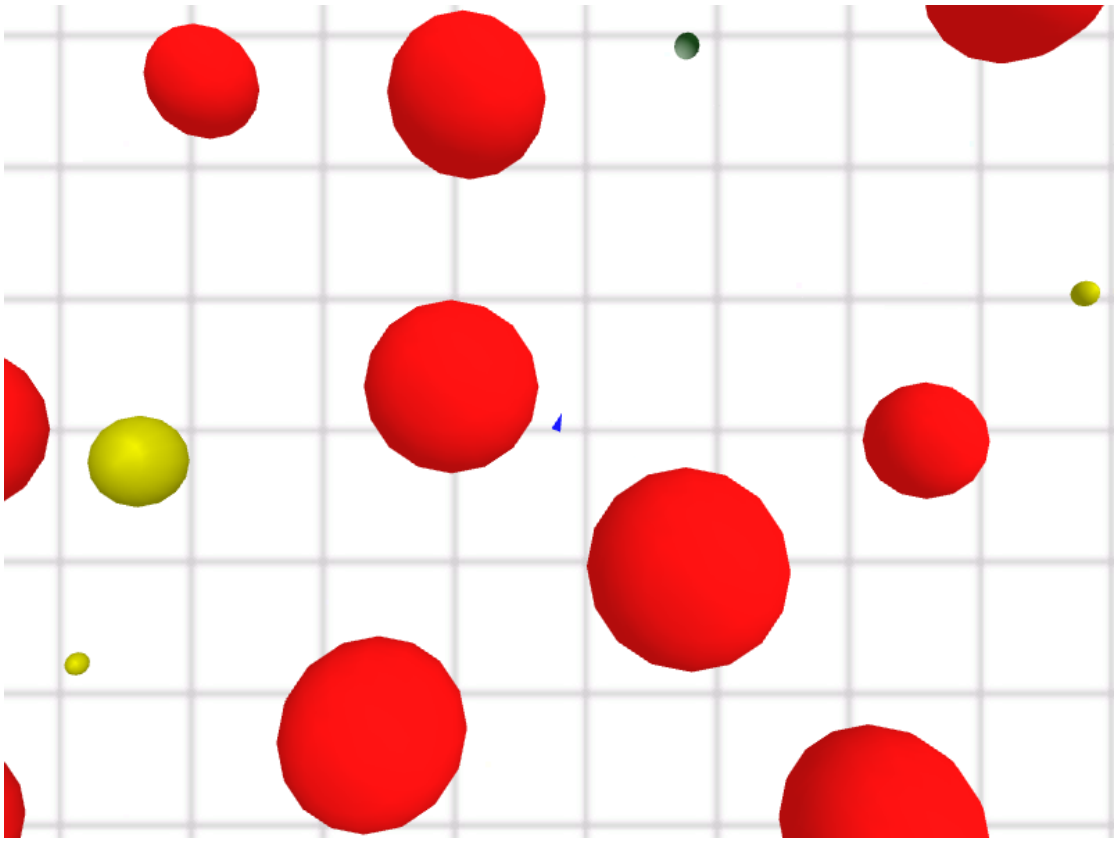


Figure 6.1: An example of one of the randomly generated CA-GS-RA environments used in experiments is shown. The agent is denoted by a blue triangle, obstacles are denoted by red spheres, “hazardous” objects are denoted by yellow spheres, and the goal is denoted by a green sphere. The orientation of the agent is random. The position and number of each obstacle and “hazardous” object is random within a specified area of the environment. The goal is placed ad a random distance within a specified interval from the agent.

of the parameters used to generate the evaluation environments is provided in Appendix B.

These forty environments were organized into ten folds of four environments each for use in cross-validation [14]. Eight folds were used as a training set, one fold was used as a validation set, and a final fold was used as a testing set. Both validation and testing sets were used to evaluate the generalizability of the learned controller. The exact uses of the validation and testing sets were learning algorithm-dependent and are discussed in Sections 6.4 and 6.5.

Each experiment consisted for forty individual runs initialized with a different random seed. Four experimental runs for each of the ten folds were performed. The same set of environments was shared between all experiments for a given primitive or composite task, while folds had different sets of training, validation, and testing environments. For example, all experiments using the two-dimensional CA-GS composite task shared the same set of environments, regardless of how the controller was developed or the architecture it used.

Agents were given a maximum of 1,500 time steps in each environment which constituted a single training episode. This was ample time for even the most risk-averse agent(s) to reach the goal location, if applicable, or to gain sufficient experience in the environment. While most of the primitive tasks used are non-episodic (the exception being the GOALSEEK primitive task), training was broken into episodes as a consequence of the nature of the evaluation environments and the primitive tasks themselves. Since the environments were unbounded, it was possible that in exploring the state space, agents could wander away from the finite number of obstacles or the other agents on the team and never have a realistic opportunity to return to the more "interesting" states of the environment. Furthermore, since much of this work is intended to operate on real robots, agent collisions warranted early termination of a training episode. Training episodes also ended early when an agent or the team of agents reached the goal location since the GOALSEEK task is inherently episodic.

Some tasks required modifications to the environments for effective controllers to be

Table 6.1: The different state abstractions used in the development of a composite behavior using the GOALSEEK primitive task are shown.

Abstraction Level	State Information	States	Total States
Full	Goal arrival time	5	175
	Goal direction Θ	7	
	Goal direction Φ	5	
Large	Goal arrival time	5	125
	Goal direction $\lvert\Theta\rvert$	5	
	Goal direction $\lvert\Phi\rvert$	5	
Medium	Goal arrival time	5	25
	Goal direction $\max(\lvert\Theta\rvert, \lvert\Phi\rvert)$	5	
Small	Goal seek priority	5	5

learned. For example, in environments used to learn the COLLISIONAVOIDANCE primitive task, obstacles randomly wandered through the environment, otherwise the agent could simply avoid collisions by not moving. A number of modifications to the environments using the FLOCKING composite task were necessary to promote learning of the desired behavior in architectures which did not use fuzzy behavior hierarchies. These included giving agents a random, initial velocity and preventing agents from coming to a full stop. Furthermore, an obstacle that moved towards the initial position of the agent was added to the environment. These modifications were made in response to the learning of sub-optimal behavior in the monolithic controllers such as immediately stopping and not moving for the duration of the episode. For reasons discussed later (see Section 7.4), these modifications were not required for architectures using adaptive fuzzy behavior hierarchies.

6.2 State Space Abstraction

Since it is possible that composite behaviors in fuzzy behavior hierarchies do not require the full state space for effective coordination of lower-level behaviors, five different levels of abstraction of the agent's state space were used when learning composite behaviors.

These were used to evaluate how abstractions affected both the rate at which effective composite behaviors were learned and their quality. Table 6.1 details the effects of each abstraction level on the state information for a composite behavior using the GOALSEEK primitive task. Figure 6.2 provides an example of each abstraction level when used with grammatical evolution.

Full This state space represents the original, joint state space of all the primitive tasks used in the composite task without any abstraction and acts as a baseline for comparison (see Figure 6.2a).

Large In this state space, state information describing directions, such as `SMALL_LEFT` or `SMALL_RIGHT`, are abstracted away into variables which denote the absolute value of the angle, such as `SMALL` (see Figure 6.2b). State information not describing a direction remains unchanged.

Medium In this state space, state information describing three-dimensional directions are abstracted away into a single variable which denotes the absolute value of the largest angle (see Figure 6.2c). State information not describing a direction remains unchanged. Note that in two-dimensional environments, this state space is the same as the **Large** state space.

Small In this state space, state information is abstracted into a single dynamic priority which is calculated using all the relevant state information local to each primitive task (see Figure 6.2d). This dynamic priority represented the task's determination of its applicability to the agent's current state. For example, using this state space, the CA-GS-RA composite behavior would only use dynamic priorities for the primitive behaviors COLLISIONAVOIDANCE, GOALSEEK, and RUNAWAY to determine how to weight its sub-behaviors. While this level of abstraction may appear to be too extreme, we have previously shown that rulesets using dynamic priorities can be developed which have similar performance to those using the **Full** state space [23].

Table 6.2: The reward functions used in developing controllers for composite tasks for each primitive task are shown. Note that while most rewards were given at each timestep, rewards for the terminal events of a collision and reaching the goal location are one-time rewards.

Primitive Task	**Description**	**Value**
COLLISIONAVOIDANCE	Collision event	-150
GOALSEEK	Goal reached event	150
	Goal distance penalty	$-0.03 \times Dist$
RUNAWAY	RunAway strength penalty	$-0.06 \times Str$
ALIGNMENT	Velocity heading difference penalty	$-0.02 \times \Delta Dir$
COHESION	Position error penalty	$-0.04 \times Dist$
SEPARATION	Separation strength penalty	$-0.02 \times Str$

Minimal In this state space, the dynamic priorities from the **Small** state space were again used. However, instead of using the dynamic priorities for every primitive behavior, only the priorities of behaviors directly weighted by a composite behavior were used. For example, the FLOCKING-CA composite behavior would only use the dynamic priorities of the FLOCKING and COLLISIONAVOIDANCE sub-behaviors.

Note that these abstractions were only used by composite behaviors. Since primitive behaviors were responsible for producing low-level control actions, they still required the unabstracted state space relevant to their primitive task. Furthermore, since monolithic controllers were also responsible for producing low-level control actions, they required the unabstracted joint state space of the overall composite task. For more detailed information regarding the size of each composite task's state space, see Table A.3.

6.3 Reward Functions

The reward functions for each primitive task as used in the development of composite tasks are shown in Table 6.2. Except for the terminal events of a collision or reaching the goal location, each reward was given per timestep. The reward values were developed with the

⟨*goal-dir-theta*⟩ ::= goal-dir-theta(BACK_LEFT)
| goal-dir-theta(LEFT)
| goal-dir-theta(SMALL_LEFT)
| goal-dir-theta(CENTER)
| goal-dir-theta(SMALL_RIGHT)
| goal-dir-theta(RIGHT)
| goal-dir-theta(BACK_RIGHT)

⟨*goal-dir-phi*⟩ ::= goal-dir-phi(DOWN)
| goal-dir-phi(SMALL_DOWN)
| goal-dir-phi(CENTER)
| goal-dir-phi(SMALL_UP)
| goal-dir-phi(UP)

⟨*goal-arrival-time*⟩ ::= goal-arrival-time(NOW)
| goal-arrival-time(REAL_SOON)
| goal-arrival-time(SOON)
| goal-arrival-time(LONG_TIME)
| goal-arrival-time(DISTANT)

(a) Full abstraction

⟨*goal-dir-theta-abs*⟩ ::= goal-dir-theta-abs(ZERO)
| goal-dir-theta-abs(SMALL)
| goal-dir-theta-abs(MEDIUM)
| goal-dir-theta-abs(LARGE)

⟨*goal-dir-phi-abs*⟩ ::= goal-dir-phi-abs(ZERO)
| goal-dir-phi-abs(SMALL)
| goal-dir-phi-abs(MEDIUM)
| goal-dir-phi-abs(LARGE)

⟨*goal-arrival-time*⟩ ::= goal-arrival-time(NOW)
| goal-arrival-time(REAL_SOON)
| goal-arrival-time(SOON)
| goal-arrival-time(LONG_TIME)
| goal-arrival-time(DISTANT)

(b) Large abstraction

Figure 6.2: Portions of grammars for the three-dimensional GOALSEEK primitive task are shown. Each used a different level of abstraction of the state space.

⟨*goal-dir-max-abs*⟩ ::= goal-dir-max-abs(ZERO)
| goal-dir-max-abs(SMALL)
| goal-dir-max-abs(MEDIUM)
| goal-dir-max-abs(LARGE)

⟨*goal-arrival-time*⟩ ::= goal-arrival-time(NOW)
| goal-arrival-time(REAL_SOON)
| goal-arrival-time(SOON)
| goal-arrival-time(LONG_TIME)
| goal-arrival-time(DISTANT)

(c) Medium abstraction

⟨*goal-seek-priority*⟩ ::= goal-seek-priority(ZERO)
| goal-seek-priority(LOW)
| goal-seek-priority(MEDIUM)
| goal-seek-priority(HIGH)

(d) Small abstraction

Figure 6.2: Portions of grammars for the three-dimensional GOALSEEK primitive task are shown. Each used a different level of abstraction of the state space. *(continued)*

maximum number of timesteps in mind and ensured that the total undiscounted reward, which was used as a performance measure in grammatical evolution, did not create a bias towards controllers which "minimized the pain" by causing a collision as quickly as possible. While some portions of the reward function were not required (e.g., the goal distance penalty), they make the reward function more "dense" and act as progress estimators by allowing learning to make the most of each experience [51, 83]. While the addition of these unrequired components may have biased policies away from the best solution, the benefit of allowing learning to make the most of each learning experience outweighed the potential problems in complex tasks such as the ones discussed in this thesis.

When learning a policy for the COLLISIONAVOIDANCE primitive task, an additional reward of 0.1 was given for each timestep in which the agent did not have a collision. While learning would occur without this reward, its addition aided in the comparison of the policy's effectiveness as learning progressed. For some composite behaviors, modifications to the reward function were necessary. For the FLOCKING composite task, a survival reward of 0.09 was given for each timestep in which the agents were active, in addition to the rewards given by the primitive tasks themselves. This served to explicitly reward the agents for avoiding collisions with other agents and continuing to flock. In single agent environments, the agent merely needed to reach the goal for the reward to be given. In multi-agent environments, the reward for reaching the goal location was only awarded if the mean position of the team was within a specified distance to the goal. This served to explicitly reward agents that reached the goal with the other agents in the team and not agents that left their team to reach the goal faster.

Due to the complexity of the tasks and the randomness of the environments, an optimal performance value for each task would be prohibitive to calculate. As a result, the performance of learned controllers cannot be compared to the performance of an optimal controller. However, based on an understanding of the experimental configuration and experience with the tasks themselves, we can identify the approximate mean performance

Table 6.3: The approximate mean performance values one would expect of an effective controller in each of the tasks are shown.

Task	**Mean Performance**	
	2D	**3D**
COLLISIONAVOIDANCE	80	140
GOALSEEK	140	140
RUNAWAY	-10	-10
CA-GS	130	140
CA-GS-RA	135	125
FLOCKING	40	40
FLOCKING-CA	0	*N/A*
FLOCKING-CA-GS	100	*N/A*
FLOCKING-CA-GS-RA	100	*N/A*

Table 6.4: Reinforcement learning parameters

Parameter	**Value**
Learning rate (α)	0.01
Discount factor (γ)	0.99
TD decay (λ)	0.25
Exploration (ε)	0.01
NN weight range	$[-0.25 : 0.25]$
NN momentum	0.01
NN hidden nodes	$1.5 \times N_{input}$

values one would expect from an effective controller. Table 6.3 details the approximate mean total reward for an effective learned controller in each of the primitive and composite tasks used.

6.4 Reinforcement Learning Configuration

As previously discussed (see Section 5.1), the Sarsa algorithm was used to learn policies for primitive and composite tasks. To speed learning, the replacing eligibility traces version of Sarsa, referred to as Sarsa(λ), was used [87]. The parameters used are shown in

Table 6.4. Since the state-action space for many of the experiments performed precluded tabular storage of the state-action values, or Q-values, neural networks were used to approximate Q-values. The neural networks consisted of a single hidden layer in which the number of nodes was a function of the number of input nodes. Unlike previous work, we found that a relatively large number of hidden nodes (1.5 times the number of input nodes) were required for policies operating in our environments [77]. Previous work in the field has concluded that using a single network to approximate all the Q-values can result in unintentional modifications of the Q-values for actions other than the one chosen by the learning algorithm [46, 77]. As a result, a separate network for each action was used in an effort to isolate the Q-values of each action.

The effects of a multi-agent environment further complicates learning as it makes the environment non-stationary [11]. To simplify the process as much as possible, we chose to use the naïve approach in which all agents used and update the same set of Q-values, or, more specifically, the same set of neural networks approximating Q-values. Experiments have shown that this form of cooperation does not impede learning and can even improve the learning rate [16, 90].

Although the use of fuzzy logic is important to providing smooth transitions between different actions in an adaptive fuzzy behavior hierarchy, fuzzy logic was not used for the behaviors developed using reinforcement learning. Therefore, primitive and composite behaviors created by reinforcement learning were not able to smoothly transition between different motor control or weighting actions, respectively, and could have potentially exhibited lower performance that controllers which do use fuzzy logic. Note that the absence of fuzzy logic in the learned policies does not prevent the use of an adaptive behavior hierarchy or fuzzy logic in the other behaviors in the hierarchy; it only has the potential to limit its effectiveness. Tunstel even comments that behaviors can be implemented using techniques other than fuzzy control if their crisp outputs are converted to fuzzy sets, which our architecture does [93].

While there are techniques for using fuzzy logic with reinforcement learning [3, 22, 27, 39], experience has shown that such modifications can increase the time needed to learn effective policies. While, in general, the increase is not prohibitive, the large number of experimental runs used for this work precluded the combination of fuzzy logic and reinforcement learning. While there are plans to use fuzzy logic with reinforcement learning (see Chapter 8), we were still able to gather conclusive results without its use (see Chapter 7).

Policies were learned using environments in the training sets. At regular intervals, the current Q-values were saved for later processing. Once learning had finished, the saved Q-values for each interval were evaluated in environments from the validation and testing sets. Results detailing progress in learning the associated task use rewards from the validation set of environments. For each experimental run, the policy with the highest reward on the validation set of environments for the entire run was chosen as the "best-of-run" policy. This policy was then evaluated on the environments in the testing set. The results of evaluating this "best-of-run" policy on the testing set are used in Chapter 7 to emphasize the learning of generalized control policies.

6.5 Grammatical Evolution Configuration

As previously discussed (see Section 5.2), grammatical evolution was used to learn policies for composite tasks. The parameters that were used are shown in Table 6.5 and are fairly normal for grammatical evolution experiments. Although initial work in evolving rulesets for adaptive fuzzy behavior hierarchies used relatively small populations of 10–20 individuals [95], a population size of 50 was used for these experiments due to the increased complexity of the tasks given to the agents. The ECJ [47] library was used to perform evolutionary runs and was modified to allow for the use of grammatical evolution.

For each composite task, grammars corresponding to each abstraction of the state space where created (see Figure 6.2). Each individual in the population was converted to a fuzzy

Table 6.5: Grammatical evolution parameters

Parameter	**Value**
Population	50
Generations	50
Codon size (bits)	8
Minimum codons	50
Maximum codons	200
Tournament size	7
Crossover type	One-point
Crossover probability	100%
Elite cloning probability	0%
Mutation probability per bit	1%

ruleset before evaluation. As a result, the controllers produced by grammatical evolution used fuzzy control, unlike composite reinforcement learning controllers which received crisp inputs and produced crisp outputs. As a result, the controllers produced by grammatical evolution have the potential for higher theoretical performance since they are able to leverage fuzzy control's ability to smoothly transition between different behavior weightings.

Evolved rulesets were evaluated in the training set of environments to determine the ruleset's fitness. Solutions that had the highest fitness on the training set, referred to as the "Best of Generation," were also evaluated in the validation set of environments. The solution with the highest fitness on the validation set over the entire run, referred to as the "Best of Run," was evaluated in the testing set of environments. This cross-validation was performed to determine the ability of the evolved rulesets to generalize to environments not encountered during training. Note that only the training fitness is used during evolution in selection for crossover. The validation and testing fitness values are only used after the evolutionary run for analysis purposes.

CHAPTER 7

Results and Discussion

As discussed in Chapter 1, this dissertation provides a number of contributions. The results, analysis, and discussion provided in this chapter provide the empirical evidence that lies at the foundations of these contributions.

- As far as we know, these experiments provide the only results of an empirical investigation into the effects of state and action abstraction for composite tasks comprised of concurrent, interfering, and non-episodic (CINE) tasks.
- The results of these experiments demonstrate that action abstraction provides more benefits than state abstraction for the development of effective agent controllers for complex tasks comprised of CINE tasks (see Section 7.4.1).
- These results also demonstrate that modular reinforcement learning, one of the few reinforcement learning approaches capable of handling CINE tasks, was not capable of learning and converging to effective policies in the problem domains used in these experiments (see Section 7.4.5).
- These results also demonstrate that composite reinforcement learning can learn effective policies for agent control in the problem domains used (see Sections 7.2.1 and 7.3.1).
- The success of composite reinforcement learning also demonstrates that behaviors designed for one task can be reused, without modification, in a new composite task (see Section 7.4.3).

- As previously discussed, the original behavior modulation algorithm does not properly handle behavior hierarchies using more than one level of composite behaviors. The results of these experiments demonstrate that this deficiency can have significant negative effects on the performance of the behavior hierarchy and that our extension successfully addresses this deficiency (see Section 7.4.2).

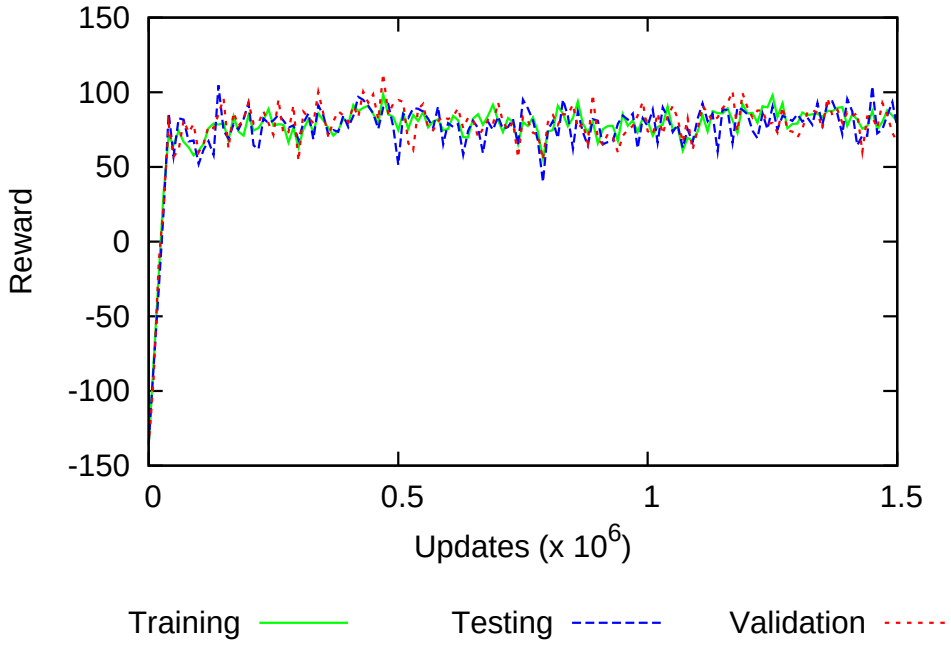
- These experiments also show that grammatical evolution is capable of evolving fuzzy rulesets for use in adaptive fuzzy behavior hierarchies (see Sections 7.2.2 and 7.3.2).

- Lastly, these results show that adaptive priorities, which provide significant state abstraction, can be used in composite behaviors to provide effective modulation of lower-level behaviors (see Sections 7.2.1 and 7.3.1).

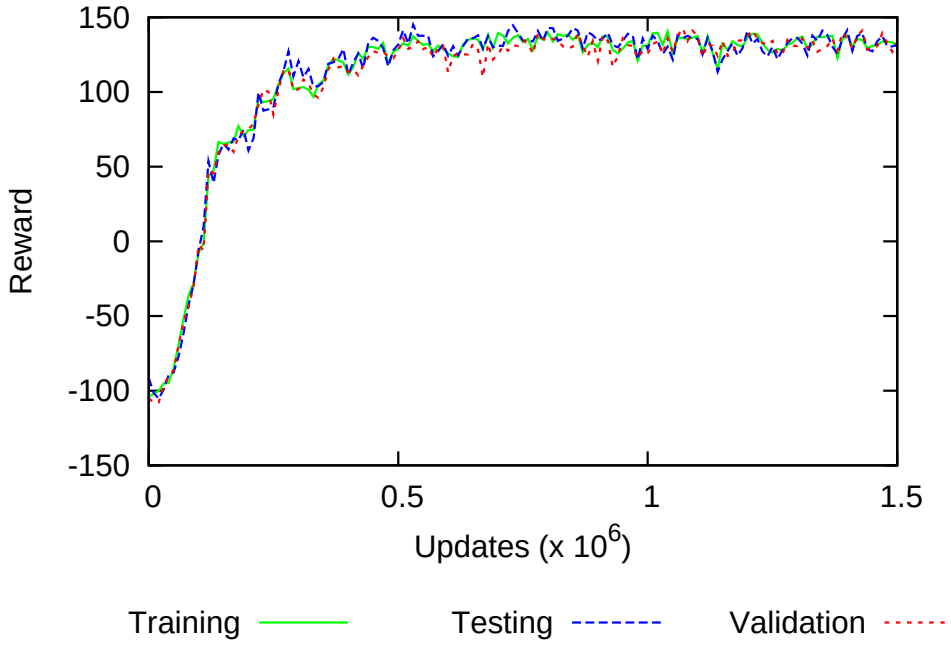
Figures showing the results of experimental runs reflect the mean performance of controllers on the validation set of environments. As stated in Section 6.1, environments were organized into ten folds for use with cross-validation. Each experiment consisted of four runs for each of the ten fold combinations for a total of 40 runs. The results of all the statistical tests performed on the experimental results are detailed in Appendix C.

7.1 Developing Primitive Task Controllers

Figure 7.1 depicts the results of using reinforcement learning to learn a policy for the COLLISIONAVOIDANCE primitive task in two and three dimensions. As can be seen, monolithic reinforcement learning was easily capable of learning an effective policy for both two and three dimensions. Although not immediately obvious, policies learned in the three-dimensional task should be able to achieve higher performance than those learned in the two-dimensional task. While the three-dimensional task has a more complex state-action space, it offers the agent greater freedom in avoiding collisions. In the two-dimensional task, a purely reactive agent can frequently be trapped by obstacles and not have the ability

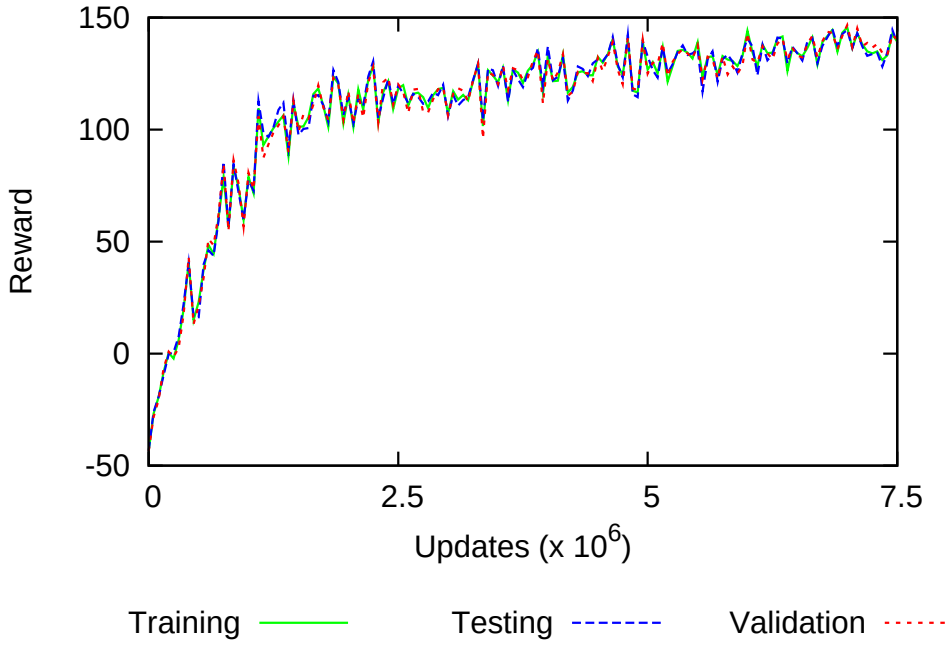


(a) 2D

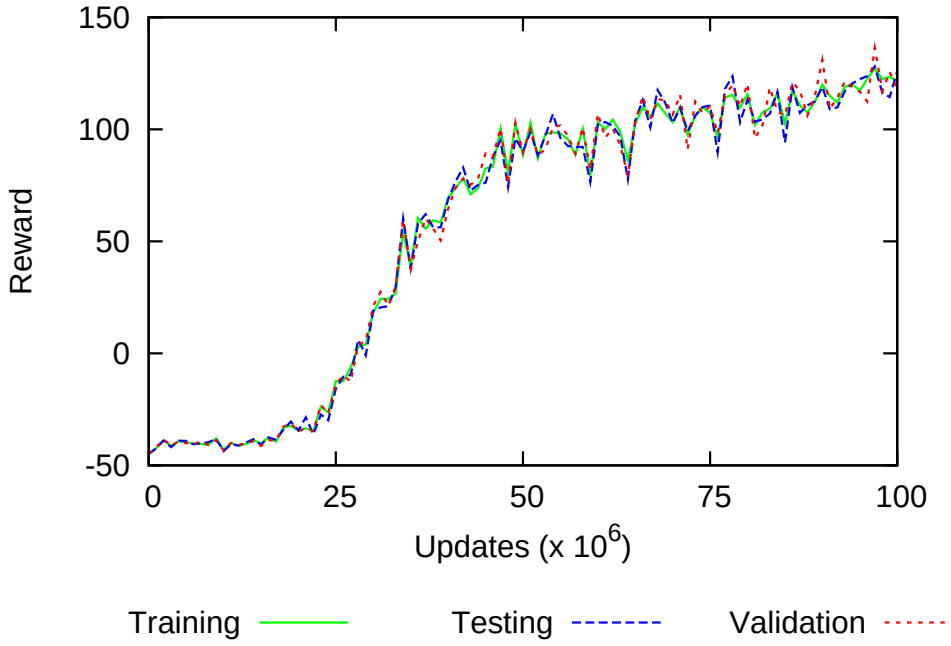


(b) 3D

Figure 7.1: Reinforcement learning results for the COLLISIONAVOIDANCE primitive task in both two and three dimensions are shown.

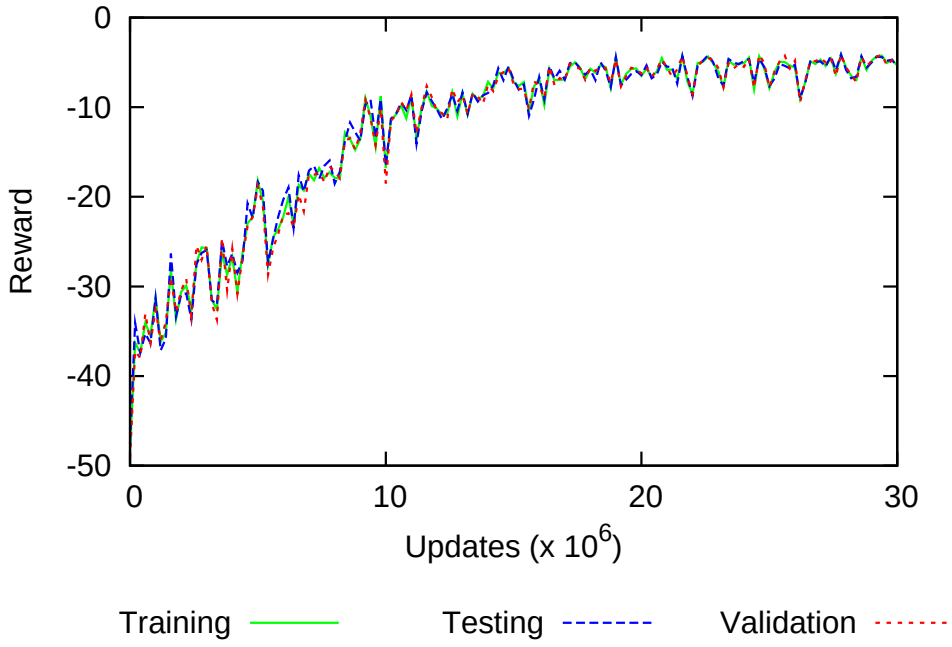


(a) 2D

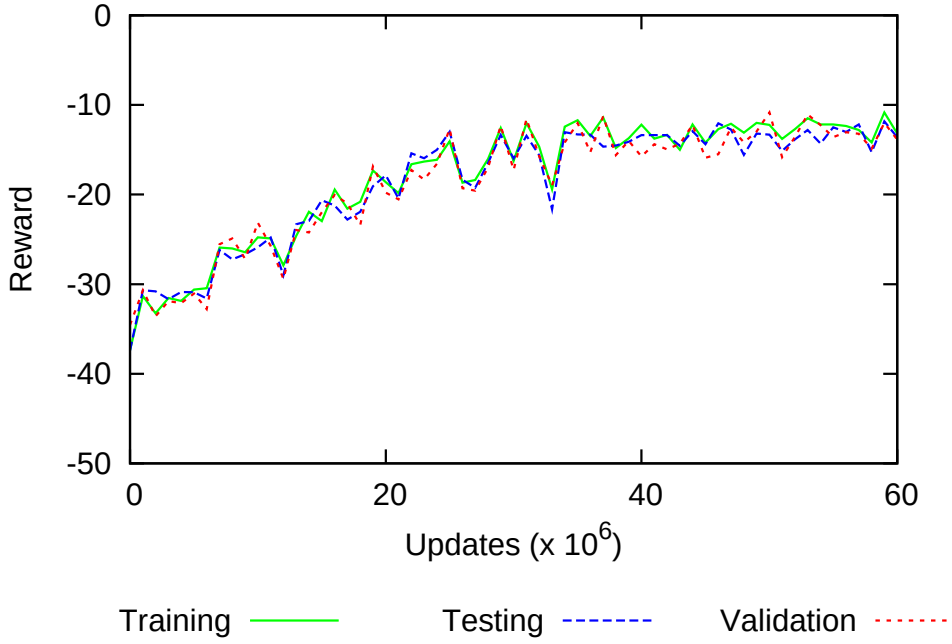


(b) 3D

Figure 7.2: Reinforcement learning results for the GOALSEEK primitive task in both two and three dimensions are shown.



(a) 2D



(b) 3D

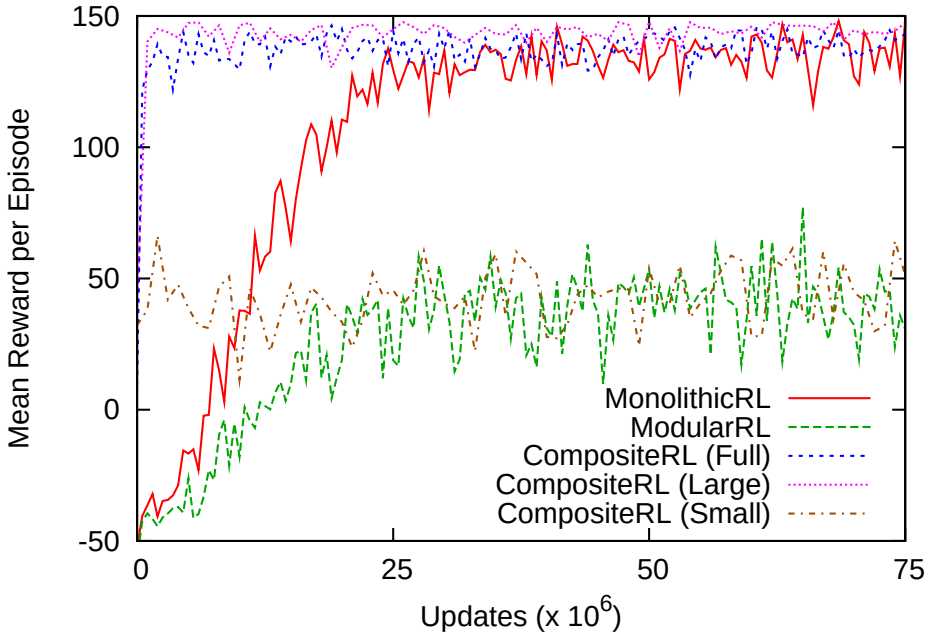
Figure 7.3: Reinforcement learning results for the RUNAWAY primitive task in both two and three dimensions are shown.

to reasonably avoid a collision. An example of an effective policy for the COLLISION-AVOIDANCE two-dimensional task can be found in Table D.1.

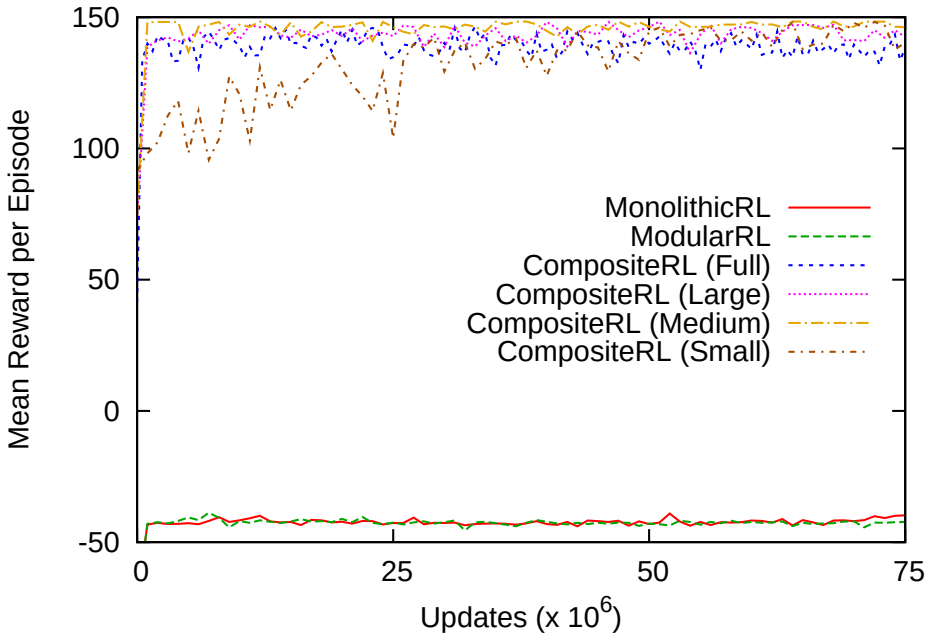
Figure 7.2 depicts the results of using reinforcement learning to learn a policy for the GOALSEEK primitive task in two and three dimensions. While the GOALSEEK primitive task can be a simpler task than the COLLISIONAVOIDANCE task, monolithic reinforcement learning was only able to gain significant traction on learning an effective policy after it has reached the goal location even with the dense reward function. Since the environments used were unbounded, it took significantly more updates to find an effective policy than for the COLLISIONAVOIDANCE primitive task. Learning effective policies for the three-dimensional task took almost two orders of magnitude longer than for the two-dimensional task and still did not achieve the same level of performance. Again, this is most likely due to the unbounded nature of the environments and the primitive task's reward function.

Figure 7.3 depicts the results of using reinforcement learning to learn a policy for the RUNAWAY primitive task in two and three dimensions. As in the previous primitive tasks, monolithic reinforcement learning was able to learn an effective policy relatively easily. An interesting result is that policies learned for the three-dimensional task had lower performance than those learned for the two-dimensional task, just as in the GOALSEEK task.

It is important to note that the policies learned for each of the primitive tasks were general and not over-specialized to the training environments. This can be seen by the approximately equal performance in the training, testing, and validation environment sets. This ability to generalize is an important prerequisite for the type of behavior reuse used by adaptive fuzzy behavior hierarchies.



(a) 2D



(b) 3D

Figure 7.4: Reinforcement learning results on the validation set environments for the CA-GS composite task in both two and three dimensions are shown.

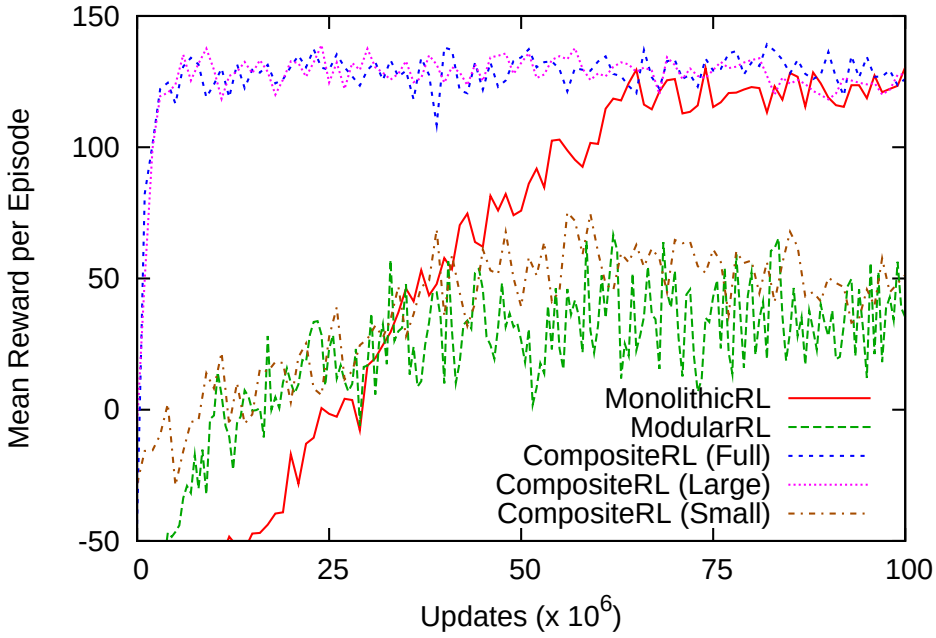
7.2 Developing Single-Agent, Composite Task Controllers

7.2.1 Reinforcement Learning

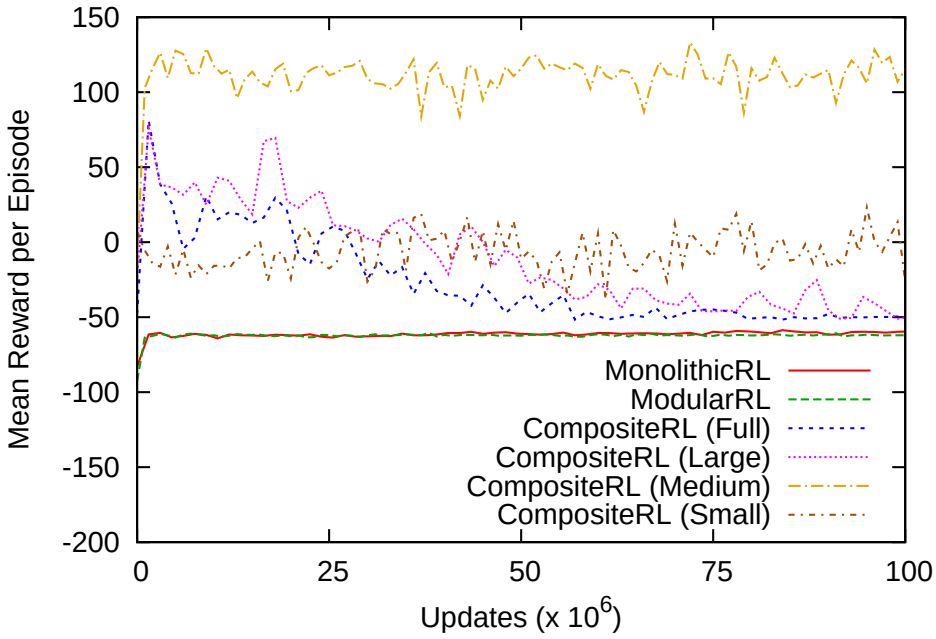
Figure 7.4 shows the results of using reinforcement learning to learn a policy for the CA-GS composite task in two and three dimensions. In general, composite reinforcement learning was able to achieve high performance within just a few updates. However, in the two-dimensional task, the **Small** abstraction level was unable to converge to an effective policy. Note that this erratic behavior was not present in the three-dimensional task nor with any of the other abstraction levels used by composite reinforcement learning. An example of an effective policy for the CA-GS composite task using the **Small** abstraction level is shown in Table D.2.

While monolithic reinforcement learning was able to learn an effective policy for the two-dimensional task, results of a randomized two-way ANOVA test demonstrates that there was a statistically significantly difference between the rate at which effective controllers were learned between monolithic and composite reinforcement learning at the 95% confidence level (see Tables C.1 and C.13 for more information). Furthermore, monolithic reinforcement learning was incapable of making any significant traction in learning an effective policy in a practical number of updates for the three-dimensional task. In both the two and three-dimensional tasks, modular reinforcement learning was unable to converge to an effective policy. In the two-dimensional task, modular reinforcement learning was able to achieve moderate success early and learned a number of effective policies, but was unable to converge to one. In the three-dimensional task, modular reinforcement learning was unable to make any significant traction, just as with monolithic reinforcement learning. While not shown here, further experiments allowed modular reinforcement learning to learn for a total of 2×10^8 updates and it was still unable to make any progress.

Figure 7.5 depicts the results of using reinforcement learning to learn a policy for the CA-GS-RA composite task in two and three dimensions. The results for the two-



(a) 2D



(b) 3D

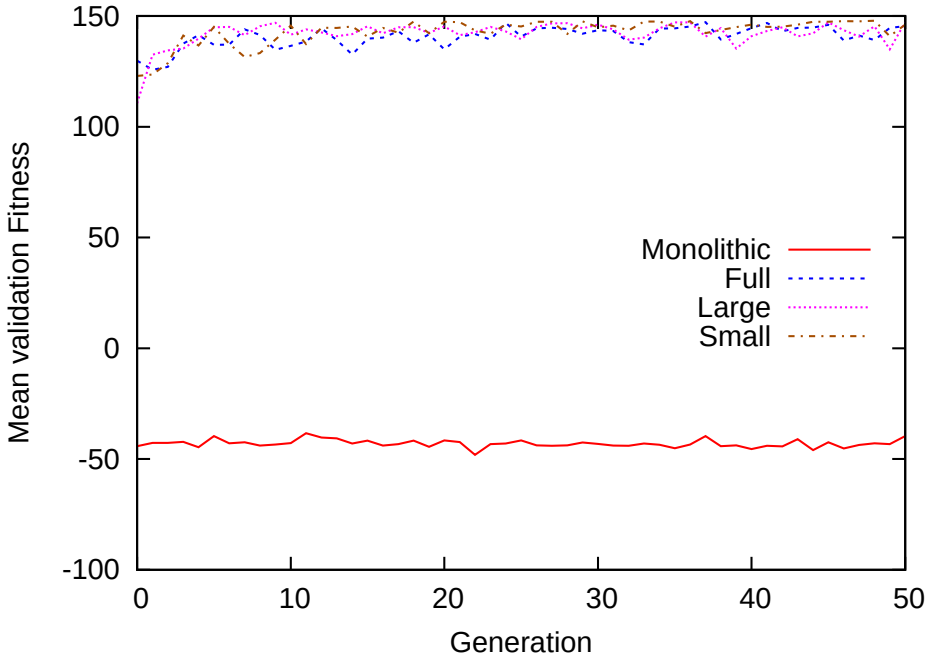
Figure 7.5: Reinforcement learning results on the validation set environments for the CA-GS-RA composite task in both two and three dimensions are shown.

dimensional environments were very similar to those of the CA-GS composite task. In general, composite reinforcement learning was able to make significant progress early in training. Again, experimental runs using the **Small** abstraction level were unable to converge to an effective policy for the two-dimensional task. However, unlike the CA-GS composite task, composite reinforcement learning had significant difficulty in learning effective controllers for abstraction levels other than **Medium**. While we believe that this loss in performance is directly attributable to a combination of the increase in the number of the state space variables and to an increase in the complexity of the environment itself, controllers learned for multi-agent composite tasks with similar properties did not exhibit the same behavior and a deeper investigation is warranted, but is tangential to the focus of this dissertation.

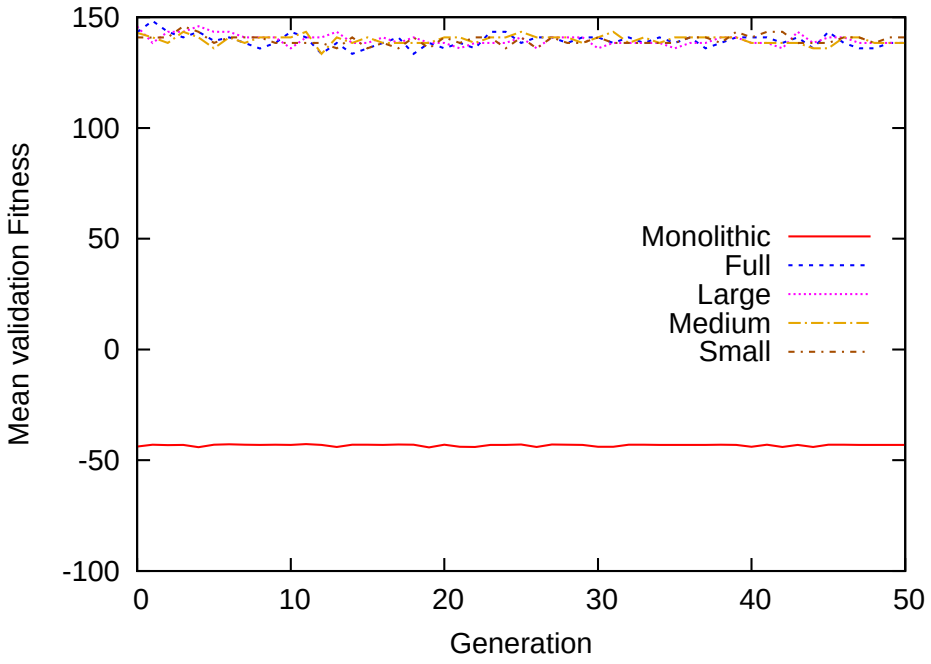
As in the CA-GS task, monolithic reinforcement learning was eventually able to learn an effective policy for the two-dimensional CA-GS-RA task after statistically significantly more updates than composite reinforcement learning at the 95% confidence level (see Table C.15), but was not able to learn an effective policy in the three-dimensional version. Modular reinforcement learning was again able to achieve moderate success in the two-dimensional task, but was unable to gain any traction in the three-dimensional task. It should be noted that in the two-dimensional task, modular reinforcement learning was capable of learning an effective policy, but was unable to converge to one.

7.2.2 Grammatical Evolution

Figure 7.6 depicts the results of using grammatical evolution to evolve a fuzzy ruleset for the CA-GS composite task in two and three dimensions. Just as in the reinforcement learning experimental runs, effective controllers using adaptive fuzzy behavior hierarchies were developed faster than monolithic controllers. Unlike the reinforcement learning runs, however, grammatical evolution was unable to evolve an effective monolithic ruleset for even the two-dimensional task. Although the fitness of the monolithic controllers indicate

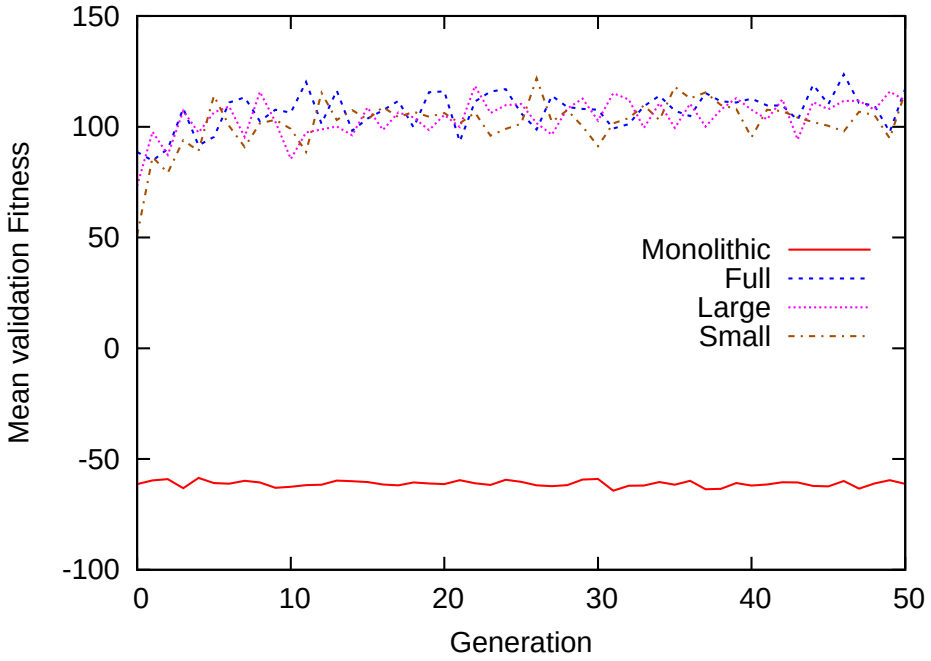


(a) 2D

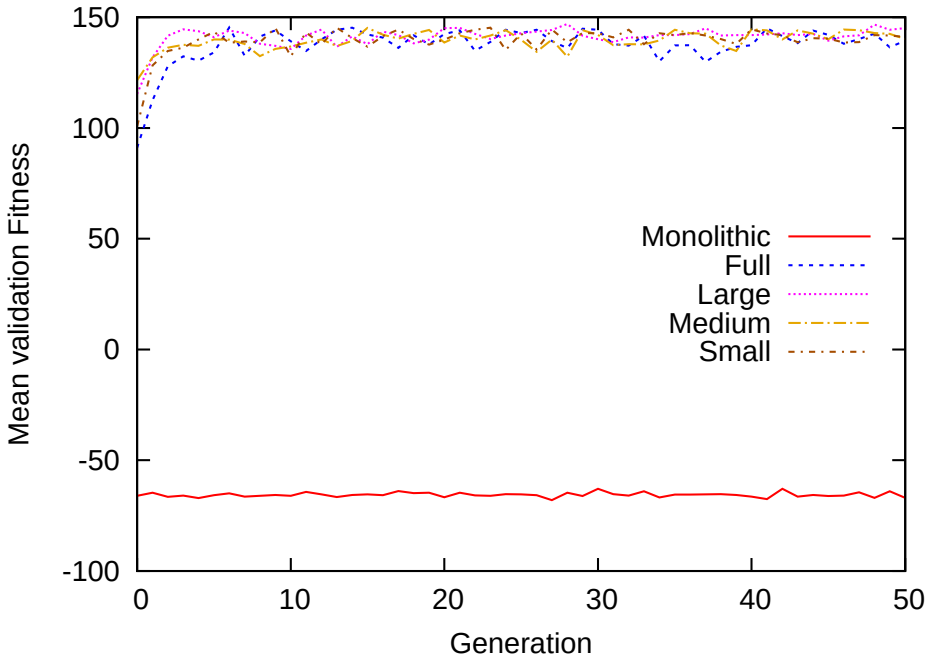


(b) 3D

Figure 7.6: Grammatical evolution results on the validation set environments for the CA-GS composite task in both two and three dimensions are shown.



(a) 2D



(b) 3D

Figure 7.7: Grammatical evolution results on the validation set environments for the CA-GS-RA composite task in both two and three dimensions are shown.

that they were able to avoid collisions, a cursory inspection of the evolved rulesets show that the agents simply do not move. Since obstacles are stationary in all environments other than those created for learning the COLLISIONAVOIDANCE primitive task itself, a simple, yet sub-optimal policy is to not move for the duration of the training episode.

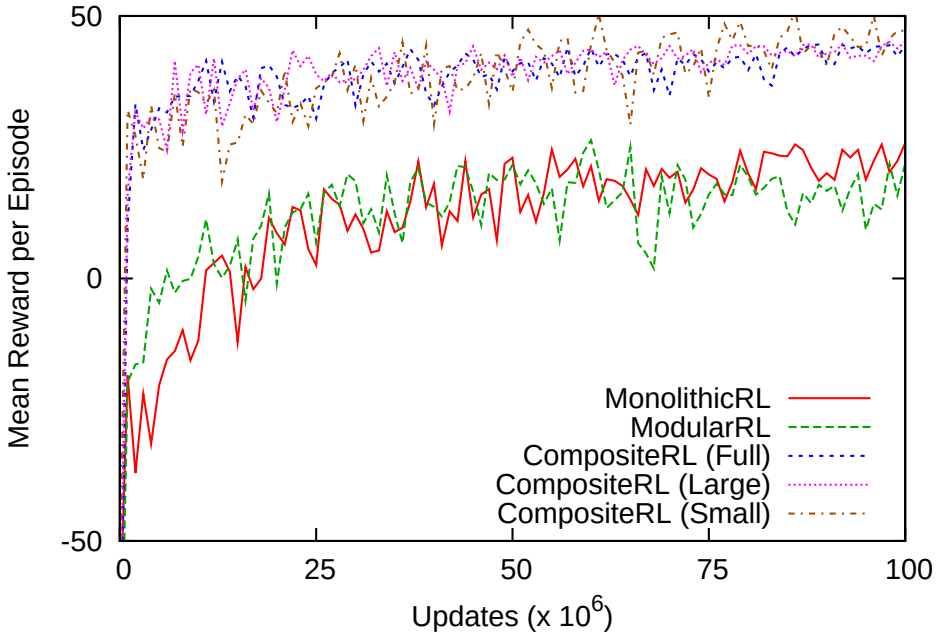
Figure 7.7 depicts the results of using grammatical evolution to evolve a fuzzy ruleset for the CA-GS-RA composite task in two and three dimensions. These results are mainly similar to those of the CA-GS composite task with only a few differences. In a number of cases, the mean best of generation fitness values in the two-dimensional version of the CA-GS-RA task were statistically significantly lower than those in the three-dimensional version at the 95% confidence level (see Table C.5 and Table C.17). These difference were not observed in the reinforcement learning experimental runs. Further inspection reveals that a number of the "Best of Run" rulesets were effective and had high fitness, but had difficulty generalizing to the validation set of environments. The reason for this loss in generalizability as compared to "Best of Run" rulesets from the other tasks is not known and merits further investigation, but is tangential to the focus of this dissertation.

Two interesting results from the evolutionary runs for both composite tasks are worthy of note. First, the erratic controller performance observed in the reinforcement learning runs using the **Small** abstraction level are not present in controllers evolved for either the CA-GS or CA-GS-RA tasks. Second, the mean best of generation fitness of rulesets for composite behaviors in the initial, random population for both the CA-GS and CA-GS-RA three-dimensional tasks were higher than those of the two-dimensional version.

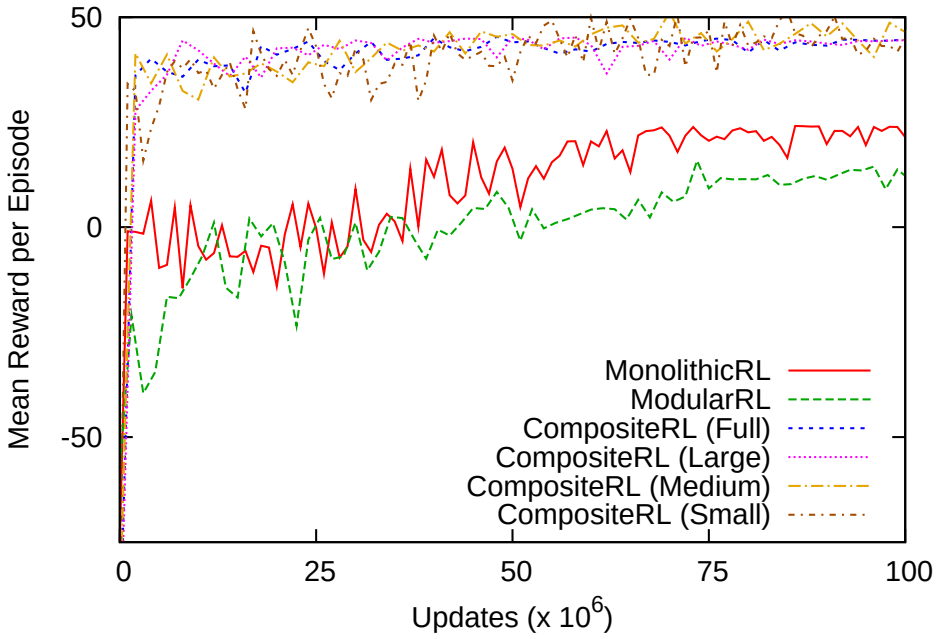
7.3 Developing Multi-Agent, Composite Task Controllers

7.3.1 Reinforcement Learning

Figure 7.8 depicts the results of using reinforcement learning to learn a policy for the FLOCKING composite task in two and three dimensions. In both types of environments,



(a) 2D



(b) 3D

Figure 7.8: Reinforcement learning results on the validation set environments for the FLOCKING composite task in both two and three dimensions are shown.

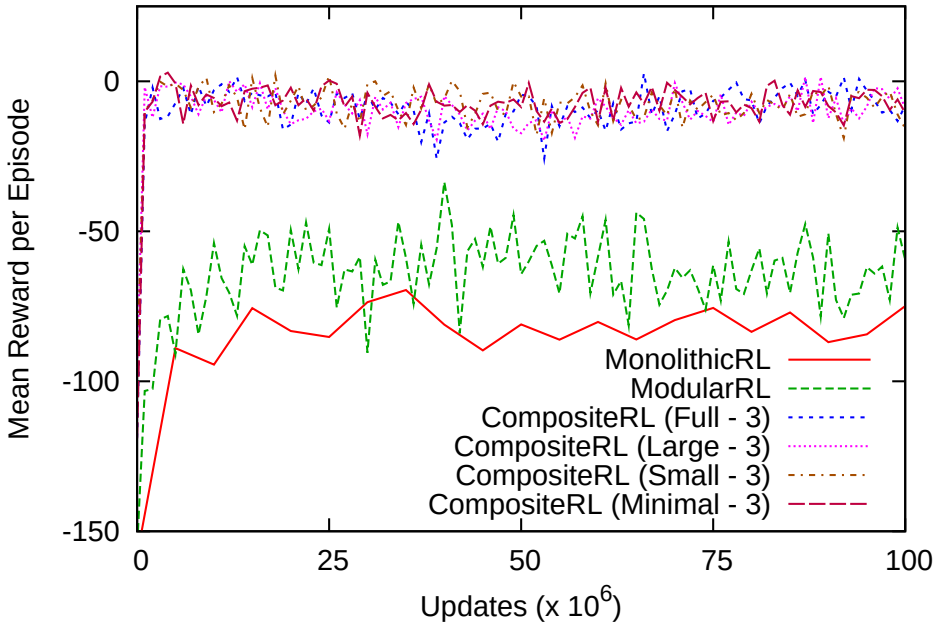


Figure 7.9: Reinforcement learning results on the validation set environments for the FLOCKING-CA composite task in two dimensions are shown. Due to storage constraints, results for monolithic reinforcement learning experimental runs used fewer checkpoints than other experimental runs.

controllers developed using the adaptive fuzzy behavior hierarchy and composite reinforcement learning had statistically significantly higher performance than those developed using either monolithic or modular reinforcement learning at the 95% confidence level (see Tables C.6 and C.18). Furthermore, controllers using the **Small** abstraction level did not exhibit the poor performance previously seen in the single agent tasks. In fact, controllers using the **Small** abstraction level had statistically significantly better performance than all other controllers in the testing set environments at the 99% confidence level as determined by the paired Student's t-test using the Bonferroni adjustment (see Table C.6).

Figure 7.9 depicts the results of using reinforcement learning to learn a policy for the FLOCKING-CA composite task in two dimensions. Again, controllers that used the adaptive fuzzy behavior hierarchy were learned faster than those that didn't use the hierarchy and had higher performance. Table D.3 details a sample policy learned using the **Minimal** abstraction level. Controllers developed using modular reinforcement learning performed

statistically significantly better than those developed using monolithic reinforcement learning at the 99% confidence level (see Table C.8), but were unable to generalize to the full range of environments present in the validation or testing sets.

Not shown in the figure are the results for controllers using a two-level adaptive fuzzy behavior hierarchy. Even with the use of the adaptive fuzzy behavior hierarchy, the significantly larger action space of the 2-level hierarchy negated any benefits that the hierarchy offered. While the hierarchy and the reuse of existing primitive behaviors meant that it did not need to learn the low-level actions needed to accomplish FLOCKING-CA, the size of the state-action space was simply too large to quickly find an effective policy. Furthermore, the increased complexity resulted in almost a four-fold increase in the wall-clock time required to learn and update Q-values for the two-level hierarchies over that of the three-level hierarchies.

Experiments using the three-dimensional environments were not performed for the FLOCKING-CA composite task or the remaining tasks due to computational resource restrictions. First, each of the forty experimental runs for a single abstraction level in the two-dimensional FLOCKING-CA task required approximately three days of computational time on the Sooner computing cluster. Experience with the three-dimensional environments in the single-agent composite tasks demonstrated that experiments using the three-dimensional environments required approximately two times the computational time as those using two-dimensional environments. As a result, almost 1,440 days of computation would be required for the three-dimensional FLOCKING-CA composite task alone. Second, the set of experiments for a single abstraction level in the two-dimensional FLOCKING-CA composite task could require over 800MB of storage. For experiments using three-dimensional environments, the storage requirements could easily exceed 4GB per set of experiments. In light of these details, and the fact that the two-dimensional results for these composite tasks already demonstrate the effects of state and action abstraction, the decision was made to not perform the three-dimensional experiments.

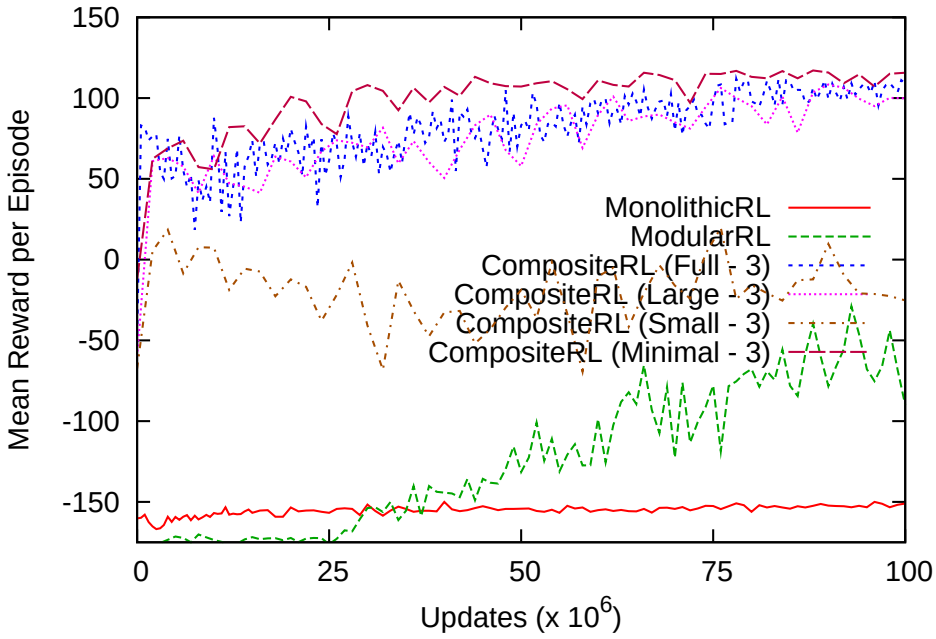


Figure 7.10: Reinforcement learning results on the validation set environments for the FLOCKING-CA-GS composite task in two dimensions are shown. The "3" denotes controllers using a three-level behavior hierarchy. The results of controllers using two-level hierarchies are not shown to improve clarity.

Figure 7.10 depicts the results of using reinforcement learning to learn a policy for the FLOCKING-CA-GS composite task in two dimensions. Just as in the previous experiments, controllers learned using composite reinforcement learning and using the adaptive fuzzy behavior hierarchy had a statistically significantly higher "best-of-run" performance than monolithic reinforcement learning at the 99% confidence level (see Table C.10) as monolithic reinforcement learning was even unable to learn how to simply avoid a collision. Controllers developed using modular reinforcement learning were able to gain some traction in learning an effective controller and were able to find effective policies. However, it was unable to converge to an effective policy and performed statistically significantly worse than composite reinforcement learning at the 95% confidence level (see Table C.20). Note that just as in the single-agent composite tasks, controllers developed using the **Small** abstraction level performed statistically significantly worse than other controllers developed using the adaptive fuzzy behavior hierarchy, including those using the **Minimal** abstraction

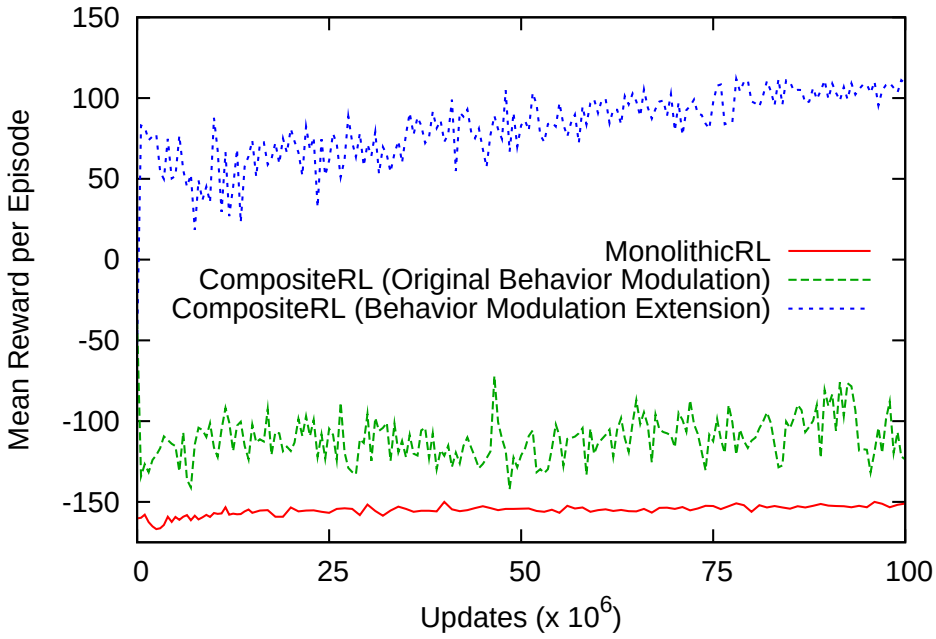


Figure 7.11: Reinforcement learning results on the validation set environments for the FLOCKING-CA-GS composite task using the **Full** abstraction level which compare the original algorithm for behavior modulation and the algorithm used in the extension are shown.

level which also uses adaptive priorities (see Table C.20).

This is the first set of results in which a difference between controllers using the different abstraction levels can be observed. Results of two-way randomized ANOVA tests show that controllers using the **Minimal** abstraction level were able to more statistically significantly achieve consistently higher performance than controllers using the other abstraction levels at the 95% confidence level (see Table C.20). However, controllers using the **Full** abstraction level were able to learn a policy at some point during learning that had statistically significantly better performance than controllers using the **Minimal** abstraction level at the 99% confidence level (see Table C.10).

Figure 7.11 depicts the results of using reinforcement learning to learn policies for the FLOCKING-CA-GS composite task using the original implementation of behavior modulation and our extension (Section 3.3) at the **Full** abstraction level. Monolithic reinforcement learning results are also provided for comparison. As can be observed, controllers using

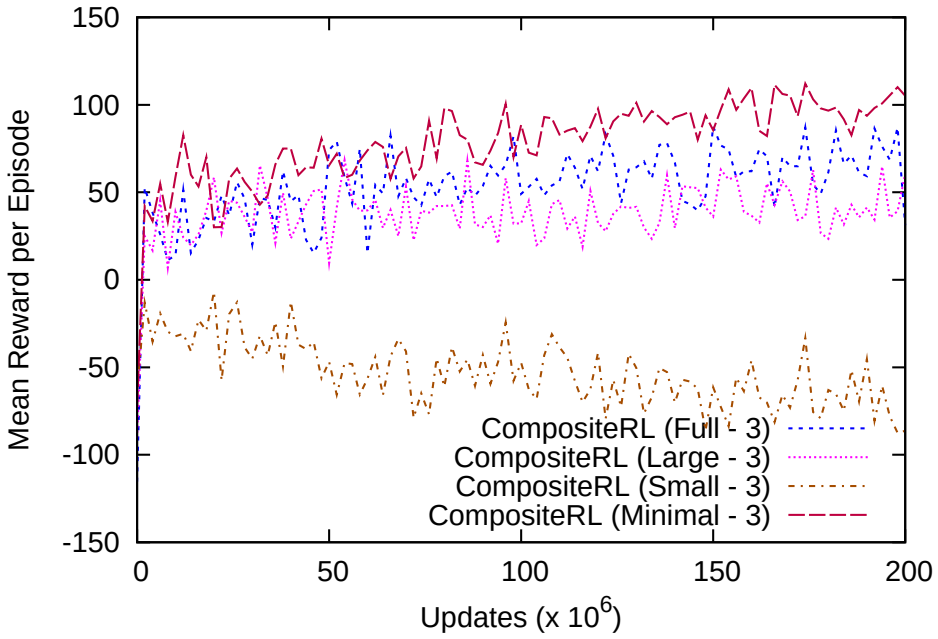


Figure 7.12: Reinforcement learning results on the validation set environments for the FLOCKING-CA-GS-RA composite task in two dimensions are shown.

the original implementation of behavior modulation only perform marginally better than controllers using a monolithic policy. However, controllers using the exact same behavior hierarchy and abstraction level, but our extension to behavior modulation, are able to have statistically significantly higher performance at the 95% confidence level (see Tables C.10 and C.20).

Figure 7.12 depicts the results of using reinforcement learning to learn a policy for the FLOCKING-CA-GS-RA composite task in two dimensions. These results detail, for the first time, a clear separation in the performance of controllers using the various abstraction levels. Randomized two-way ANOVA tests show that, like the FLOCKING-CA-GS composite task, controllers using the **Minimal** abstraction level were able to more statistically significantly achieve consistently higher performance than controllers using the other abstraction levels at the 95% confidence level (see Table C.21), but there was no statistically significant difference in the "best-of-run" testing fitness between controllers using the different abstraction levels (see Table C.12). Controllers using the **Full** abstraction level had

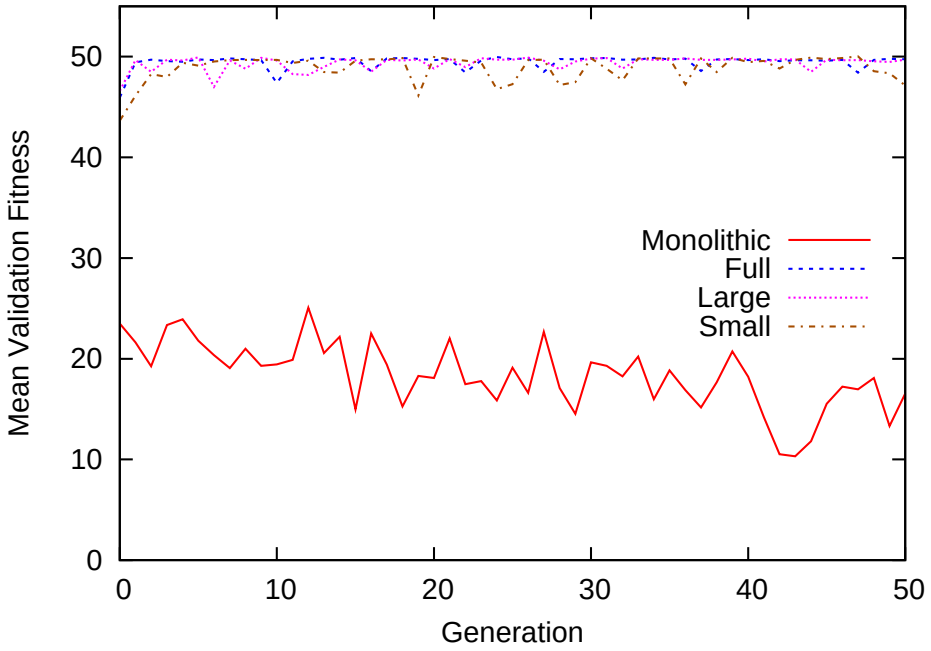


Figure 7.13: Grammatical evolution results on the validation set environments for the FLOCKING composite task in two dimensions are shown.

a higher mean reward per episode than controllers using the **Small** abstraction level at the 95% confidence level (see Table C.12). Monolithic and modular reinforcement learning were not used to learn controllers due to their consistent poor performance in the simpler, multi-agent composite tasks.

7.3.2 Grammatical Evolution

Figure 7.13 depicts the results of using grammatical evolution to evolve a fuzzy ruleset for the FLOCKING composite task in two dimensions. Although the fitness values indicate that the evolved rulesets were quite effective in coordinating the primitive behaviors, an inspection of the actual rulesets reveals otherwise. In nearly every ruleset inspected, the only primitive behavior given a non-zero weight was the ALIGNMENT primitive behavior. In a few rulesets, the SEPARATION was also given a non-zero weight. While the focus on the ALIGNMENT primitive behavior meant that the ruleset was able to maintain a FLOCKING formation that resulted in high fitness, it also meant that the ruleset was not

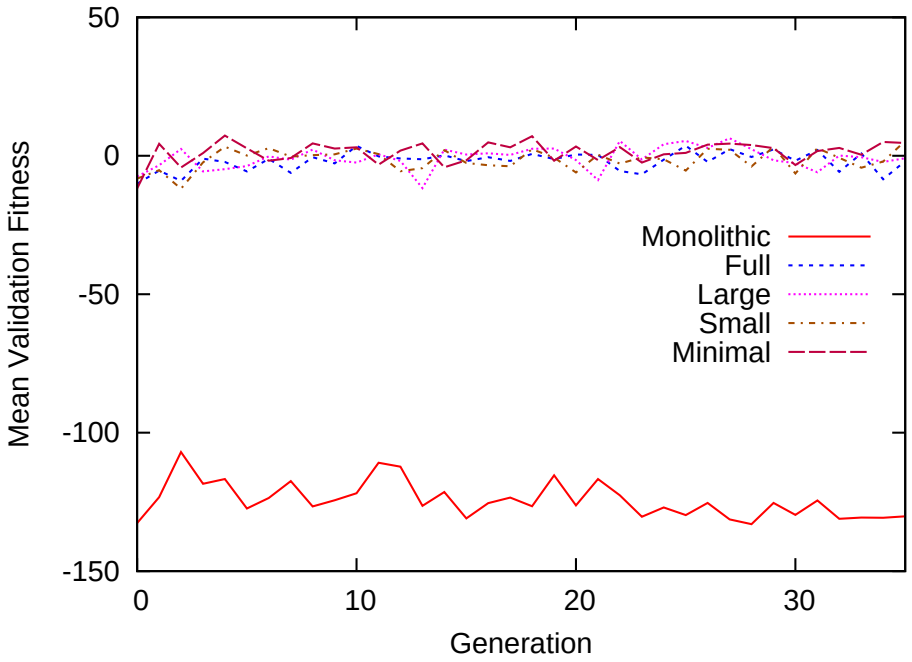


Figure 7.14: Grammatical evolution results on the validation set environments for the FLOCKING-CA composite task in two dimensions are shown.

an effective general solution. Its use would be inappropriate in more complex tasks in which COHESION and SEPARATION would be necessary to re-form a stable formation that could be maintained with ALIGNMENT. There was no statistically significant difference between "best-of-run" controllers for the **Full**, **Large**, or **Small** abstraction levels using the Student's paired t-test (see Table C.7).

Figure 7.14 depicts the results of using grammatical evolution to evolve a fuzzy ruleset for the FLOCKING-CA composite task in two dimensions. Evolved controllers using an adaptive fuzzy behavior hierarchy significantly outperformed those evolved using a monolithic fuzzy ruleset. There was no statistically significant difference between "best-of-run" controllers developed using either the **Full**, **Large**, **Small**, or **Minimal** abstraction levels using the Student's paired t-test (see Table C.9).

Figure 7.15 depicts the results of using grammatical evolution to evolve a fuzzy ruleset for the FLOCKING-CA-GS composite task in two dimensions. As with the FLOCKING rulesets produced by grammatical evolution, the fitness values of the evolved FLOCKING-

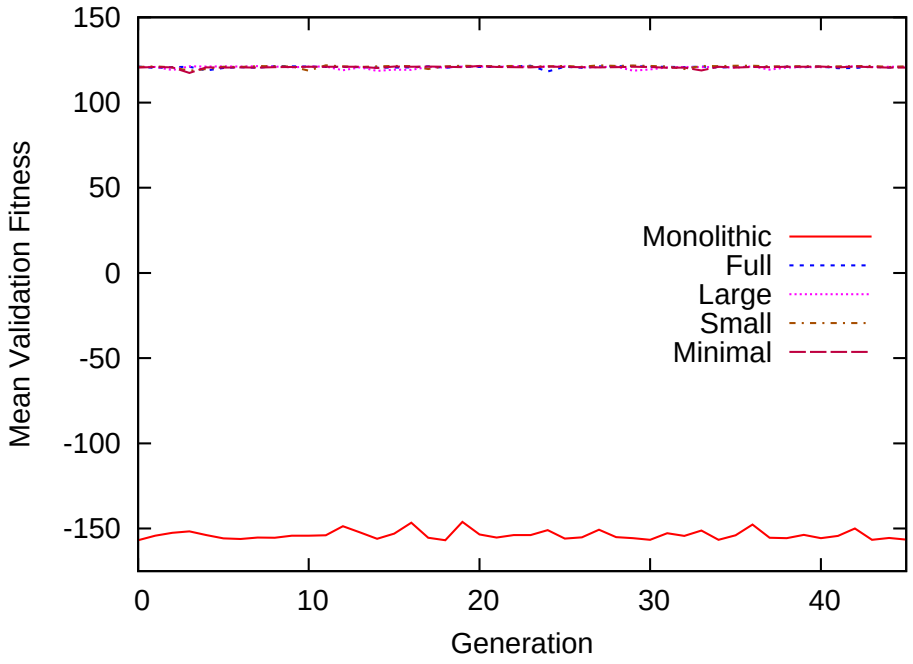


Figure 7.15: Grammatical evolution results on the validation set environments for the FLOCKING-CA-GS composite task in two dimensions are shown.

CA-GS rulesets indicate that the controllers were effective, while an inspection of the rulesets reveals otherwise. The "best-of-run" rulesets consistently gave only the CA-GS composite behavior a non-zero weight and ignored the FLOCKING composite behavior (see Figure 4.5c). As a result, each of the agents independently navigated towards the goal location. Although the reward for reaching the goal location was only awarded when the entire team reach the goal, the agents still received the reward since all the agents eventually were within range of the goal location. There was no statistically significant difference between the "best-of-run" controllers using the **Full**, **Large**, **Small**, or **Minimal** abstraction levels using the Student's paired t-test (see Table C.11).

7.4 Analysis and Discussion

These results demonstrate that controllers using adaptive fuzzy behavior hierarchies significantly outperformed controllers with other architectures in terms of performance, rate of

development, or both. This performance difference can most clearly be seen in the three-dimensional versions of the composite tasks where monolithic controllers and controllers using modular reinforcement learning were unable to provide any measure of effective control.

7.4.1 State and Action Abstraction

While there are many reasons for this improvement, we believe that the central reason for this improvement is the action abstraction that adaptive fuzzy behavior hierarchies provide. Note that in the two-dimensional CA-GS-RA task, the action space for monolithic controllers consisted of only two variables: the change in speed and the change of direction. However, for controllers using composite behaviors, the action space of the composite behavior consisted of three variables: the weights for the COLLISIONAVOIDANCE, GOALSEEK, and RUNAWAY primitive behaviors. Despite this increased action space, controllers using composite behaviors were developed significantly faster and with significantly better performance in the three-dimensional tasks. This is due to the fact that although the action space of the composite behavior was larger, it consisted of high-level, abstracted "meta-actions" instead of the more complex, low-level control actions. As has been previously discussed, although other approaches also use the concept of "meta-actions," the action abstraction used in this dissertation is fundamentally different since the primitive tasks used were, in general, concurrent, interfering, and non-episodic.

We can conclude that action abstraction is more useful than state abstraction by comparing the performance of controllers used for the complex, multi-agent composite tasks. In these tasks, the controller could be designed with a hierarchy that used a single composite behavior or a hierarchy that mad use of multiple composite behaviors in different hierarchical levels. An example is the three alternatives for implementing the FLOCKING-CA-GS composite behavior shown in Figure 4.5. Results show that even with significant state abstraction, controllers using the two-level hierarchy shown in Figure 4.5a were only

able to achieve mediocre performance due to the sheer size of the action space. However, effective controllers using the three-level hierarchy shown in Figure 4.5c were able to be developed without any abstraction of the agent's state. It is important to also note that using reinforcement learning to learn controllers using the two-level hierarchy in the FLOCKING-CA-GS task took significantly more computational time than those using the three-level hierarchy. This is due to the fact that in the Sarsa algorithm, the Q-Value for each of the 3,125 possible actions must be calculated at each time step. Over the course of the entire experimental run, this resulted in almost a four fold increase in the wall clock time required to develop controllers using a two-level hierarchy over those using a three-level hierarchy.

A surprising result is that for many of the composite tasks evaluated, there was no statistically significant difference between controllers using the various state abstraction levels with the exception of the **Small** abstraction level (see Section 7.4.4). While the use of abstraction can significantly reduce the size of the state space, the possibility of over-abstracting the state space and negatively impacting the controller's performance exists. However, in general, this was not observed.

What is not reflected in these results is the computational effort required to develop the primitive behaviors used by the controllers using adaptive fuzzy behavior hierarchies. The primitive behaviors used in these experiments were manually developed and were designed to be effective, but not optimal. Since the primitive tasks associated with these behaviors are relatively simple, the process of manually creating these rulesets was straightforward. However, if manually creating the behaviors is impractical, results show that effective policies for the single-agent behaviors can be easily learned. Even when the the computational effort of creating the primitive behaviors is included, developing controllers that use adaptive fuzzy behavior hierarchies is still far more beneficial and practical than the other approaches evaluated.

7.4.2 Behavior Modulation Extension

Fundamental to the advantages of using adaptive fuzzy behavior hierarchies and the associated action abstraction is the concept of behavior modulation. However, as the results show, the original implementation of behavior modulation is incapable of producing effective control for composite tasks that require complex hierarchies. However, when the same hierarchy is used with our extension to behavior modulation, a significant increase in performance is observed. We believe that the loss of performance associated with the original implementation is a direct result of the lack of a composite behavior's activation level having an impact on the contributions of the primitive behaviors to the overall control action. In the FLOCKING-CA-GS composite task, where there are many conflicting primitive tasks, effective modulation of the primitive behaviors is essential. Without it, there is no coordination between the FLOCKING and CA-GS composite behaviors. As a result, the controller is unable to accomplish either task.

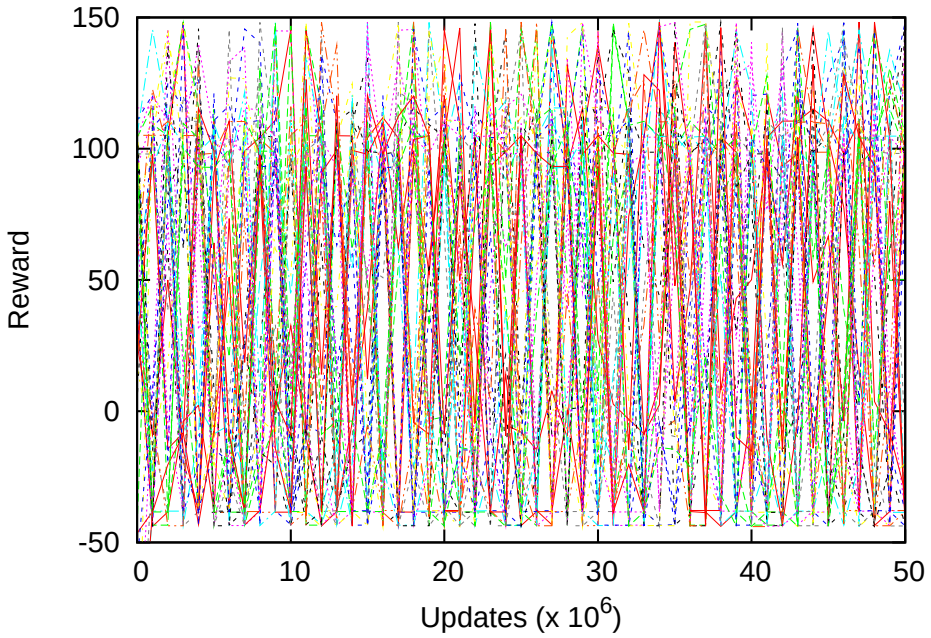
7.4.3 Behavior Reuse

In each of the experiments shown, controllers using an adaptive fuzzy behavior hierarchy reused behaviors, both primitive and composite, that had been developed for simpler tasks. Furthermore, these controllers were used without modification. The ability to reuse existing behaviors not only simplified the development effort required to produce effective controllers, but it also simplified the development process itself, further improving the practicality of developing effective controllers for complex, composite CINE tasks. While the reused behaviors in these experiments were developed manually, in the future, we wish to reuse behaviors developed using composite reinforcement learning and grammatical evolution.

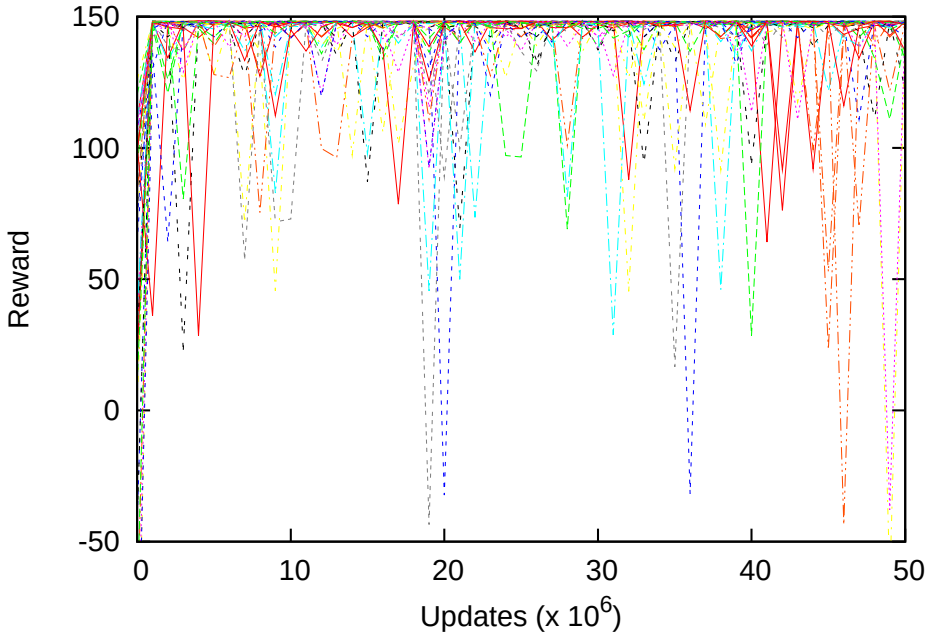
7.4.4 Performance of Controllers Using the Small Abstraction Level

The performance of controllers developed with reinforcement learning and using the **Small** abstraction level are of particular interest as it may indicate that we have over-abstracted the state space. Figure 7.16 shows the validation set reinforcement learning curves for all forty runs using both the **Small** and **Large** abstraction levels. Note that while an effective policy could be learned using the **Small** abstraction level, reinforcement learning was unable to *converge* to an effective policy, although it could when using the **Large** abstraction level. While there are a number of potential reasons for this lack of convergence, we do not believe this is an inherent problem with the dynamic priorities used by the **Small** abstraction level. There were a number of composite tasks for which controllers using the **Small** abstraction level provided effective control. In fact, effective controllers using the **Small** abstraction level for the CA-GS and CA-GS-RA composite tasks were evolved using grammatical evolution (see Figure 7.6a). We believe the root cause of the problem lies in the combination of the dynamic priority generated for the GOALSEEK primitive task and reinforcement learning. In each composite task, where the combination was required, the development of controllers using the **Small** abstraction level was unable to converge.

To evaluate our hypothesis, we altered the **Small** abstraction level by replacing the GOALSEEK adaptive priority with the GOALSEEK state information from the **Full** abstraction level (i.e., the arrival time at and direction to the goal location) and performed a number of the experiments again. The results of these experiments using the altered **Small** abstraction level were compared to the results using the original **Small** abstraction level in Figures 7.17–7.20. As can be seen, controllers that did not use the GOALSEEK adaptive priority were learned faster, had higher performance, or both when compared to controllers which did use the priority. Although not shown, the performance of controllers using the modified **Small** abstraction level were comparable to the performance of the controllers learned using the other abstraction levels. Randomized two-way ANOVA tests were performed using these new results and show that controllers using this altered

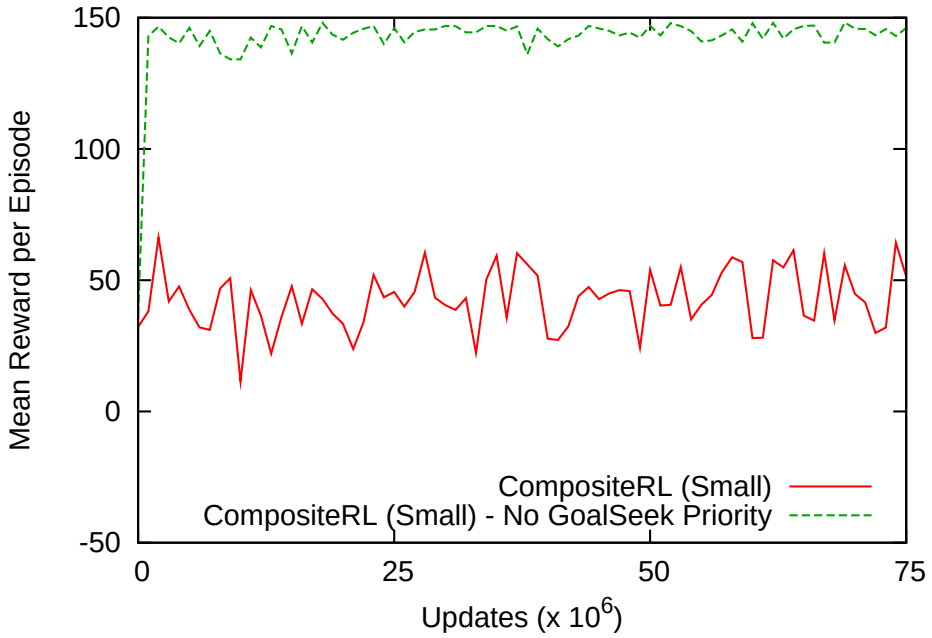


(a) Small abstraction level

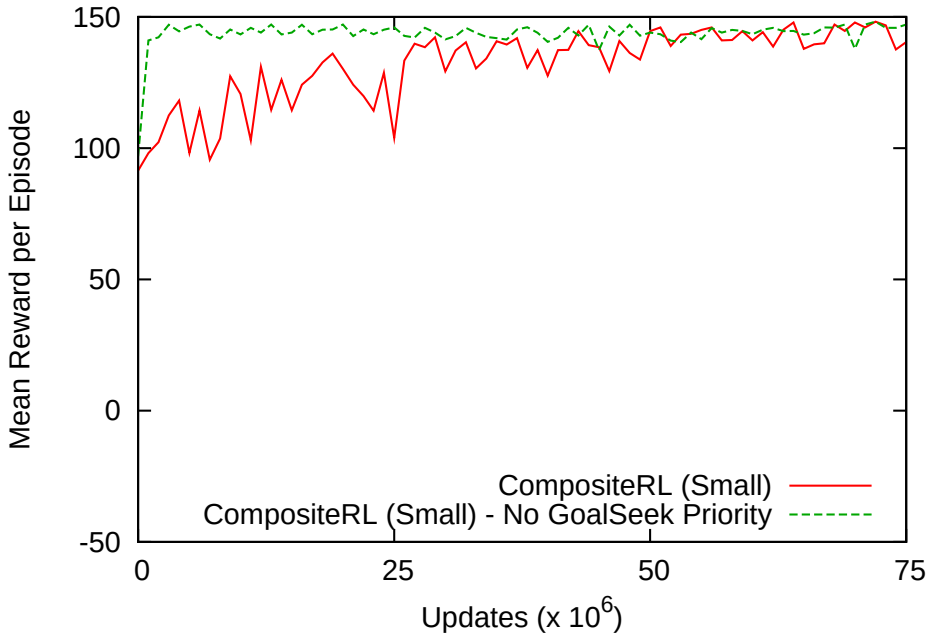


(b) Large abstraction level

Figure 7.16: Reinforcement learning results on the validation set environments for all runs learning the CA-GS composite task using the **Small** and **Large** abstraction levels. Note that controllers using the **Small** abstraction level can have high performance, but fail to converge like those using the **Large** abstraction level.

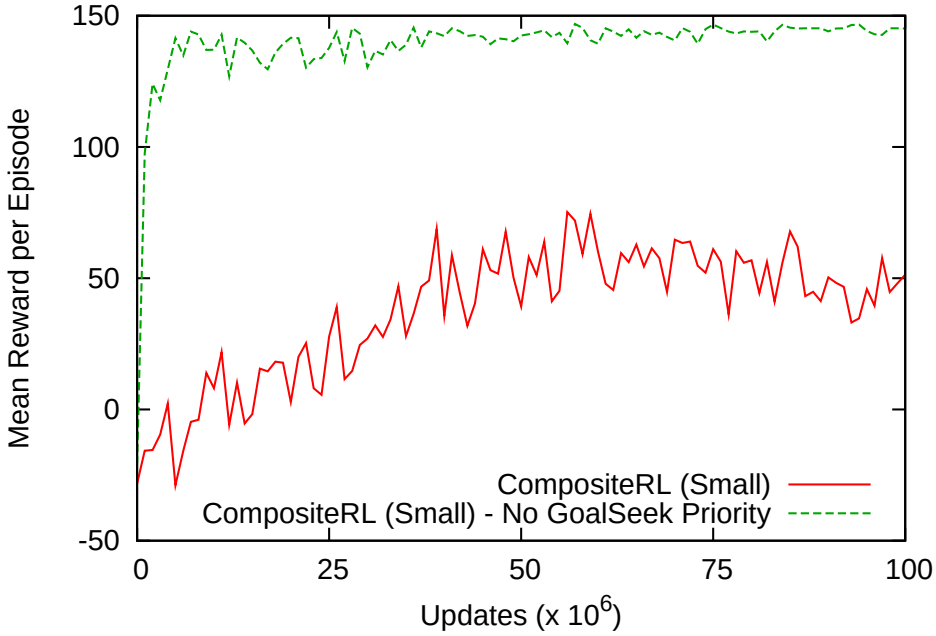


(a) 2D

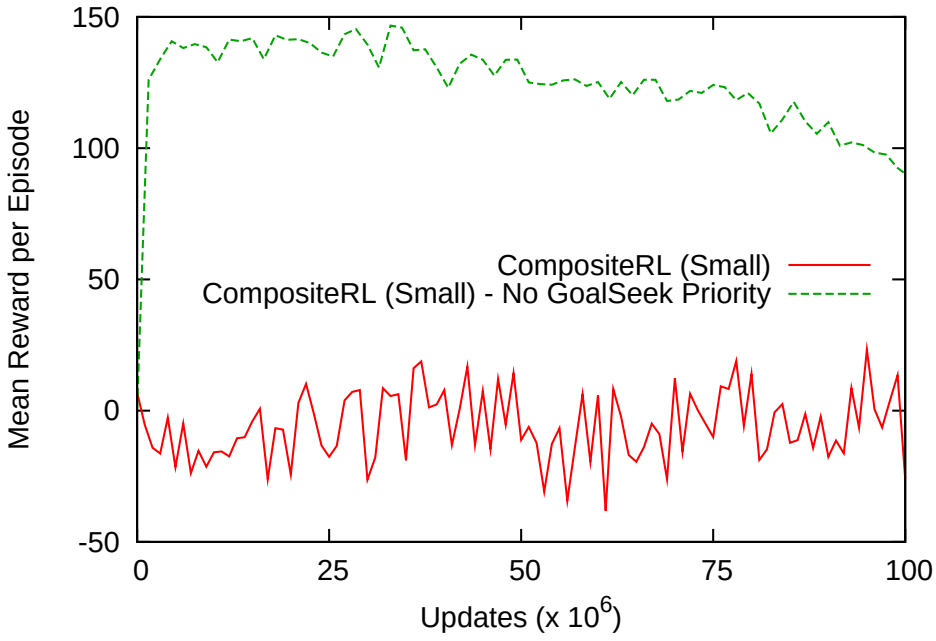


(b) 3D

Figure 7.17: Reinforcement learning results on the validation set environments for the CA-GS composite task comparing different approaches for the **Small** abstraction level in both two and three dimensions are shown.



(a) 2D



(b) 3D

Figure 7.18: Reinforcement learning results on the validation set environments for the CA-GS-RA composite task comparing different approaches for the **Small** abstraction level in both two and three dimensions are shown.

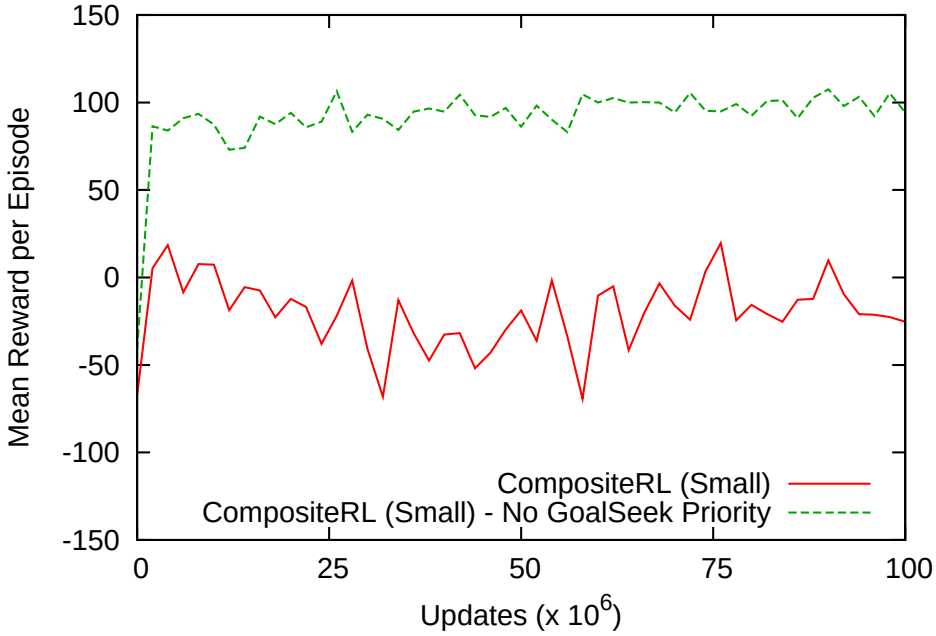


Figure 7.19: Reinforcement learning results on the validation set environments for the FLOCKING-CA-GS composite task comparing different approaches for the **Small** abstraction level in two dimensions are shown.

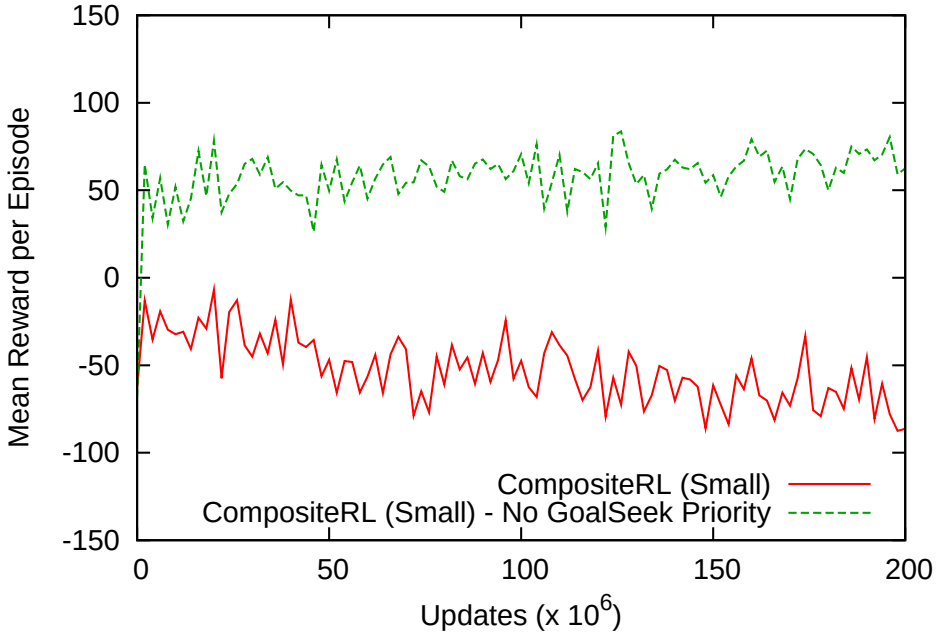


Figure 7.20: Reinforcement learning results on the validation set environments for the FLOCKING-CA-GS-RA composite task comparing different approaches for the **Small** abstraction level in two dimensions are shown.

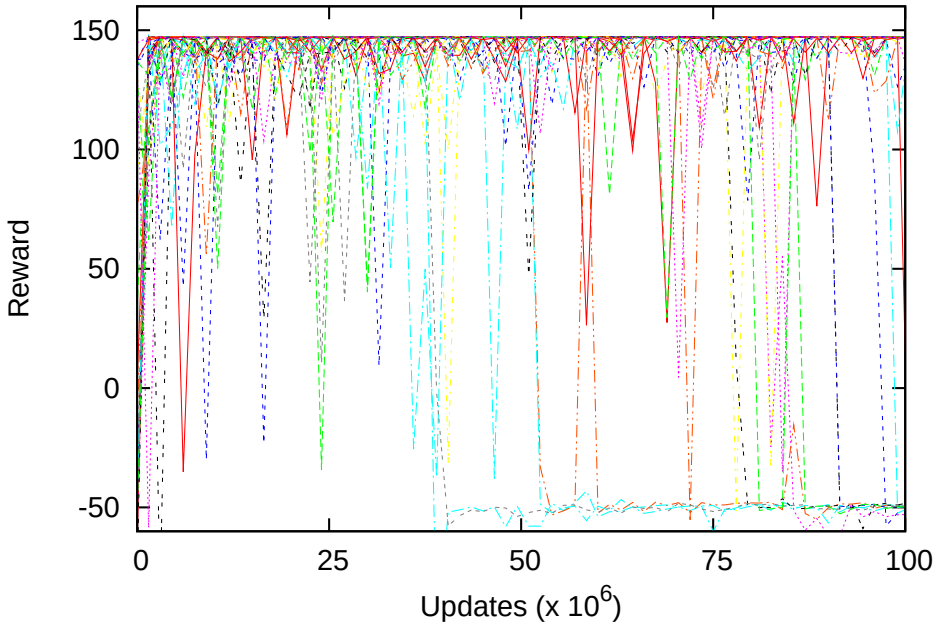


Figure 7.21: Reinforcement learning results on the validation set environments for all runs learning the CA-GS-RA composite task using the modified **Small** abstraction level in which the GOALSEEK adaptive priority was replaced with the full state information. Compare these learning curves with those found in Figure 7.16a.

Small abstraction level are consistent with the results for the other abstraction levels in the FLOCKING-CA-GS composite task and are comparable to controllers using the **Full** abstraction level in the FLOCKING-CA-GS-RA composite task. This indicates that the GOALSEEK adaptive priority was somehow the contributing factor to the low performance in the previous experiments. Exactly why the GOALSEEK adaptive priority causes such problems is unknown and worthy of future investigation, but is tangential to the focus of this dissertation.

The loss in performance for the three-dimensional CA-GS-RA controllers is perplexing as such a significant loss in performance was not observed in any other run, despite the generally poor performance of controllers using the other abstraction levels (see Figure 7.18b). Figure 7.21 indicates that only a few runs exhibited this loss in performance and that the loss started occurring around 50×10^6 updates. This coincides with the point at which exploration of the state spaces becomes increasingly rare. While this explains

the inability of reinforcement learning to re-learn the more rewarding actions, it does not explain why such re-learning would be necessary.

7.4.5 Performance of Modular Reinforcement Learning

While the results of previous work show that modular reinforcement learning can be an effective learning algorithm (see Spraque and Ballard [84] for an example), the results presented here do not confirm this. In each of the composite tasks, modular reinforcement learning was unable to provide performance that was competitive with composite reinforcement learning. In fact, in the CA-GS, CA-GS-RA, and FLOCKING composite tasks, modular reinforcement learning was only able to achieve mediocre performance at best. In analyzing the individual results for the single-agent tasks, it is apparent that the problem was not that modular reinforcement learning converged to a sub-optimal policy, but rather that it was unable to converge to a policy at all. At first, learning progressed rapidly as is described in previous work. However, after moderate success, the policy began to oscillate between consistent success and consistent failure (see Figure 7.5a). In the more complex multi-agent composite tasks, modular reinforcement learning was unable to achieve even moderate success. Although modular reinforcement learning as an algorithm is capable of producing controllers for composite CINE tasks, these experiments demonstrate that it was not a viable alternative for the tasks and state spaces used here.

7.4.6 Command Fusion Issues

While beneficial, there can be problems with using command fusion for agent control [63]. The most significant problem with command fusion is that the process of fusing two opposite actions can result in an action that is inappropriate for any primitive task. However, we were unable to find evidence of any such situations in the experiments described here. While it is possible that the implementation of the evaluation environment prevented such problems, we believe that the process of evolving a ruleset that weights the primitive be-

haviors was successful at avoiding such situations due to the low fitness of the weighting actions which can result in inappropriate actions.

7.4.7 Development of Desired Behavior

By using a hierarchy built with effective primitive behaviors, developing controllers that exhibited the desired behavior was significantly easier than the alternative methods which did not reuse existing behaviors. As previously discussed, significant modifications to the FLOCKING and FLOCKING-CA environments were necessary in order for monolithic and modular reinforcement learning to learn controllers that even came close to the desired behavior and not learn to simply stop. Since simply stopping is not a valid action for any of the primitive behaviors in those environments, controllers developed using the adaptive fuzzy behavior hierarchy and composite reinforcement learning did not require any such modifications to produce the desired behavior. As a result, the process of developing an effective reward function can be simplified without compromising the learned policy.

7.4.8 Use of Grammatical Evolution

Although grammatical evolution was able to quickly evolve effective rulesets for the single-agent tasks using the adaptive fuzzy behavior hierarchy, experiments using the multi-agent tasks had mixed results. While the rulesets evolved for the FLOCKING composite task, they were clearly inappropriate for use in environments other than those used for evolution as they relied solely on the ALIGNMENT primitive behavior. Such was not the case for the behaviors produced by composite reinforcement learning. A similar situation was observed in the FLOCKING-CA-GS composite task where the evolved rulesets relied only on the CA-GS-RA composite behavior and completely ignored FLOCKING. While significant alterations to both the training environments and the fitness functions could be made to produce the desired behavior, such alterations were not necessary for controllers learned using composite reinforcement learning. While there were benefits in using grammatical

evolution over composite reinforcement learning (e.g., the lack of a loss in performance when using the **Small** abstraction level and less computation time), grammatical evolution's heightened sensitivity to both the training environments and the fitness function mean that considerably more manual effort must be used to produce controllers of comparable performance to those of composite reinforcement learning.

CHAPTER 8

Conclusions and Future Work

8.1 Conclusions

The experiments and results described in this dissertation reflect the only empirical investigation of which we are aware that directly compares the effects of state and action abstraction on the development of controllers for composite tasks comprised of concurrent, interfering, and non-episodic primitive tasks. Development of controllers for these types of composite tasks have received comparatively little attention, especially in the machine learning community, and yet are tasks that are frequently encountered in the development of autonomous agents.

The most significant result of these experiments is that, in the problem domains used in this dissertation, the abstraction of an agent's action space provided more tangible benefits in the development of agent controllers than abstraction of an agent's state space. In a direct comparison, controllers that used significant action abstraction and no state abstraction had higher performance and were developed faster than controllers that made extensive use of state abstraction and moderate action abstraction. This is due to the fact that action abstraction changed the focus of the controller from one of low-level control to one of high-level coordination. This change in focus not only made the development of controllers for complex composite tasks more practical, but in many of the tasks, it also allowed the controller to have higher performance.

While adaptive fuzzy behavior hierarchies do provide significant benefits, experimental results showed that the original implementation of behavior modulation used in adaptive fuzzy behavior hierarchies is limited to hierarchies of two levels. When applied to more

complex hierarchies, a significant loss in performance occurred. However, the extension to behavior modulation used here removed this limitation, making adaptive fuzzy behavior hierarchies usable for far more complex composite tasks.

One aspect that is fundamental to the improved performance and rate of development of controllers using adaptive fuzzy behavior hierarchies was the ability to reuse existing primitive and composite behaviors. The ability to reuse, without modification, behaviors developed for one task in another task allowed for the development of controllers in individual pieces. The benefits of this approach are apparent when compared to the other approaches evaluated in these experiments which attempted to develop a controller all at once. As a result of this reuse, controllers for complex composite tasks that were once impractical to develop can now be developed with reasonable effort.

The results of the experiments shown in this dissertation demonstrate that while the use of modular reinforcement learning has been successful in more constrained problem domains, it was unable to consistently produce effective control policies in the problem domains used here. For the problem domains used, the prospect of simultaneously learning effective policies for each primitive task proved to be too complicated. In contrast, composite reinforcement learning was the only approach that was consistently able to produce effective control policies. As discussed above, we believe that this was due to the use of action abstraction and the ability to reuse existing primitive behaviors, regardless of their implementation.

These results also demonstrate that grammatical evolution can be used to produce effective rulesets for autonomous agent control. While evolved rulesets did not produce the desired behavior for some of the more complex multi-agent tasks, grammatical evolution did have many benefits over reinforcement learning, the most important being that it was capable of producing more effective controllers in the single-agent composite tasks in less computation time. Although grammatical evolution appears to be more sensitive to the construction of a fitness function than composite reinforcement learning, these results

show that it is fully capable of producing controllers which are just as effective as those developed using composite reinforcement learning.

8.2 Future Work

As in most research, the results of this dissertation offer more questions than they answer and introduce many opportunities for future work. The most immediate opportunity for future work is a more in-depth investigation of the erratic behavior of using the **Small** abstraction level with the GOALSEEK primitive task. The results of this could provide more insight into the state versus action abstraction comparison performed in this dissertation. Furthermore, a deeper understanding of the erratic behavior could lead to improvements in the creation and calculation of adaptive priorities with a particular emphasis on using machine learning techniques. Also, an investigation of the learned policies described in Appendix D should provide more insight into why there is relatively little variance in the control policies for the composite behaviors.

The next opportunity for future work is to use fuzzy reinforcement learning to learn composite behavior policies instead of the discrete Sarsa approach [39]. The use of fuzzy reinforcement learning offers the potential for faster learning since the agent's state is no longer confined to a single discrete value. As a result, the agent is able to gain experience, at varying degrees, in multiple states simultaneously. Furthermore, fuzzy control policies offer the same smoothness of control that traditional fuzzy rulesets offer.

Lastly, the results of the current work can be used to develop more complex controllers in a variety of ways. One way is to directly use adaptive fuzzy behavior hierarchies to create far more complicated behavior hierarchies. Since our ultimate focus is in the development of agent controllers for use in complex, multi-agent tasks, the results of this work provide significant contributions in making the development of such controllers practical. Another way is to combine adaptive fuzzy behavior hierarchies with other approaches to produce more complex controllers. This can be done by introducing planning or using other reactive

approaches. One of the more promising approaches is to combine adaptive fuzzy behavior hierarchies with multi-objective behavior coordination [61]. The use of multi-objective behavior coordination avoids many of the potential pitfalls of using fuzzy command fusion while the use of adaptive fuzzy behavior hierarchies retains the ability to dynamically adapt the prioritization of the primitive tasks to changes in the environment.

Bibliography

[1] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003.

[2] Sven Behnke and Raúl Rojas. A hierarchy of reactive behaviors handles complexity. In *Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, volume 2103 of *Lecture Notes in Computer Science*, pages 125–136. Springer-Verlag, 2001.

[3] Hamid R. Berenji. A reinforcement learning–based architecture for fuzzy logic control. *International Journal of Approximate Reasoning*, 6(2):267–292, 1992.

[4] Sooraj Bhat, Charles Lee Isbell Jr., and Michael Mateas. On the difficulty of modular reinforcement learning for real-world partial programming. In *National Conference on Artificial Intelligence (AAAI)*, volume 21, pages 318–325. AAAI Press, 2006.

[5] Andrea Bonarini. Evolutionary learning of fuzzy rules: Competition and cooperation. *Fuzzy Modelling: Paradigms and Practice*, pages 265–283, 1996.

[6] Andrea Bonarini. Reinforcement learning of hierarchical fuzzy behaviors for autonomous agents. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 1223–1228, 1996.

[7] Andrea Bonarini, Giovanni Invernizzi, Thomas Halva Labella, and Matteo Matteucci. An architecture to coordinate fuzzy behaviors to control an autonomous robot. *Fuzzy Sets and Systems*, 134(1):101–115, 2003.

[8] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[9] Yuehui Chen, Bo Yang, Ajith Abraham, and Lizhi Peng. Automatic design of hierarchical takagi-sugeno type fuzzy systems using evolutionary algorithms. *IEEE Transactions on Fuzzy Systems*, 15(3):385–397, 2007.

[10] Yu-Chiun Chiou and Lawrence W. Lan. Genetic fuzzy logic controller: an iterative evolution algorithm with new encoding method. *Fuzzy Sets and Systems*, 152(3):617–635, 2005.

[11] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *National Conference on Artificial Intelligence (AAAI)*, pages 746–752, 1998.

[12] Robert Cleary. Extending grammatical evolution with attribute grammars: An application to knapsack problems. Master's thesis, University of Limerick, University of Limerick, Ireland, 2005.

[13] Robert Cleary and Michael O'Neill. An attribute grammar decoder for the 01 multiconstrained knapsack problem. *Evolutionary Computation in Combinatorial Optimization*, pages 34–45, 2005.

[14] Paul R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1995.

[15] Oscar Cordón, Fernando A. C. Gomide, Francisco Herrera, Frank Hoffmann, and Luis Magdalena. Ten years of genetic fuzzy systems: Current framework and new trends. *Fuzzy Sets and Systems*, 141(1):5–31, 2004.

[16] Robert H. Crites and Andrew G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33(2):235–262, 1998.

[17] Luiz Soares de Oliveira, Robert Sabourin, Flavio Bortolozzi, and Ching Y. Suen. A methodology for feature selection using multiobjective genetic algorithms for handwritten digit string recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 17(6):903–929, 2003.

[18] Thomas G. Dietterich. An overview of MAXQ hierarchical reinforcement learning. In *International Symposium on Abstraction, Reformulation, and Approximation*, pages 26–44. Springer-Verlag London, UK, 2000.

[19] Thomas G. Dietterich. State abstraction in MAXQ hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 994–1000, 2000.

[20] Dimiter Driankov, Hans Hellendoorn, and Michael Reinfrank. *An Introduction to Fuzzy Control*. Springer-Verlag, second edition, 1996.

[21] Meng Joo Er and Chang Deng. Online tuning of fuzzy inference systems using dynamic fuzzy q-learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(3):1478–1489, 2004.

[22] Meng Joo Er and Yi Zhou. A novel reinforcement learning approach for automatic generation of fuzzy inference systems. In *IEEE International Conference on Fuzzy Systems*, pages 100–105, Vancouver, BC, 16-21 July 2006.

[23] Brent E. Eskridge and Dean F. Hougen. Prioritizing fuzzy behaviors in multi-robot pursuit teams. In *IEEE International Conference on Fuzzy Systems*, pages 1119–1125, 16-21 July 2006.

[24] Brent E. Eskridge and Dean F. Hougen. Using priorities to simplify behavior coordination. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1334–1336, 2007.

[25] Fernando Fernández and Manuela M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 720–727, 2006.

[26] Mohammad Ghavamzadeh, Sridhar Mahadevan, and Rajbala Makar. Hierarchical multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 13(2):197–229, 2006.

[27] Pierre Yves Glorennec and Lionel Jouffe. Fuzzy Q-learning. In *IEEE International Conference on Fuzzy Systems*, volume 2, pages 659–662, 1997.

[28] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, March 2003.

[29] Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lotfi A. Zadeh. *Feature Extraction: Foundations and Applications*. Studies in Fuzziness and Soft Computing. Springer-Verlag, Secaucus, NJ, USA, 2006.

[30] Bernhard Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *International Conference on Machine Learning*, pages 243–250. Morgan Kaufmann, 2002.

[31] Frank Hoffmann. An overview on soft computing in behavior based robotics. In *International Fuzzy Systems Association World Congress*, pages 544–551, 2003.

[32] Frank Hoffmann. Fuzzy behavior coordination for robot learning from demonstration. In *International Conference of the North American Fuzzy Information Processing Society*, pages 157–162, 2004.

[33] John H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI, USA, 1975.

[34] Manfred Huber. Learning hierarchical control policies using closed-loop actions. In *IASTED International Conference on Artificial Intelligence & Soft Computing*, pages 356–361. IASTED.

[35] Manfred Huber and Roderic A. Grupen. A feedback control structure for on-line learning tasks. *Robotics and Autonomous Systems*, 22(3):303–315, 1997.

[36] Mark Humphrys. Action selection methods using reinforcement learning. In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 135–144. MIT Press, Bradford Books, 1996.

[37] Nicholas K. Jong and Peter Stone. State abstraction discovery from irrelevant state variables. In *International Joint Conference on Artificial Intelligence*, pages 752–757, August 2005.

[38] Anders Jonsson and Andrew G. Barto. Automated state abstraction for options using the U-tree algorithm. In *Advances in Neural Information Processing Systems*, pages 1054–1060. MIT Press, 2001.

[39] Lionel Jouffe. Fuzzy inference system learning by reinforcement methods. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, 28(3):338–355, 1998.

[40] Jonas Karlsson. *Learning to Solve Multiple Goals*. PhD thesis, University of Rochester, Rochester, NY, USA, 1997.

[41] Daphne Koller and Ronald Parr. Computing factored value functions for policies in structured MDPs. In *International Joint Conference on Artificial Intelligence*, pages 1332–1339. Morgan Kaufmann, 1999.

[42] George Konidaris and Andrew G. Barto. Building portable options: Skill transfer in reinforcement learning. In *International Joint Conference on Artificial Intelligence*, pages 895–900, 2007.

[43] John R. Koza. *Genetic programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[44] C. C. Lee. Fuzzy logic in control systems: Fuzzy logic controller. I. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):404–418, March 1990.

[45] C. C. Lee. Fuzzy logic in control systems: Fuzzy logic controller. II. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):419–435, March 1990.

[46] Long-Ji Lin. Scaling up reinforcement learning for robot control. In *International Conference on Machine Learning*, pages 182–189. Morgan Kaufmann, 1993.

[47] Sean Luke. ECJ 15: A Java evolutionary computation library. http://cs.gmu.edu/∼eclab/projects/ecj/, 2006.

[48] Zhihui Luo, David Bell, and Barry McCollum. Skill combination for reinforcement learning. In *Intelligent Data Engineering and Automated Learning*, volume 4881 of *Lecture Notes in Computer Science*, pages 87–96. Springer, 2007.

[49] Eric Martinson, Alexander Stoytchev, and Ronald C. Arkin. Robot behavioral selection using q-learning. In *IEEE/RSJ International Conference on Intelligent Robots and System*, volume 1, pages 970–977, 2002.

[50] Maja J. Matarić. Reward functions for accelerated learning. In *International Conference on Machine Learning*, pages 181–189, 1994.

[51] Maja J. Matarić. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):73–83, 1997.

[52] Vicente Matellán, Camino Fernández, and José M. Molina López. Genetic learning of fuzzy reactive controllers. *Robotics and Autonomous Systems*, 25(1-2):33–41, 1998.

[53] Mausam and Daniel S. Weld. Solving concurrent markov decision processes. In *National Conference on Artificial Intelligence*, pages 716–722. AAAI Press, 2004.

[54] Andrew K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1996.

[55] David J. Montana. Strongly typed genetic programming. Technical Report #7866, 10 Moulton Street, Cambridge, MA 02138, USA, 7 1993.

[56] Robin R. Murphy. *Introduction to AI Robotics*. MIT Press, Cambridge, MA, USA, 2000.

[57] Monica Nicolescu and Maja J. Matarić. A hierarchical architecture for behavior-based robots. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 227–233, Bologna, Italy, July 2002. ACM Press.

[58] Michael O'Neill and Conor Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[59] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, pages 1043–1049. Morgan Kaufmann, 1997.

[60] Justus H. Piater, Paul R. Cohen, Xiaoqin Zhang, and Michael Atighetchi. A randomized ANOVA procedure for comparing performance curves. *International Conference on Machine Learning*, pages 430–438, 1998.

[61] Paolo Pirjanian. *Multiple Objective Action Selection & Behavior Fusion using Voting*. PhD dissertation, Aalborg University, Denmark, 1998.

[62] Paolo Pirjanian. Behavior coordination mechanisms – state-of-the-art. Technical Report IRIS-99-375, Institute for Robotics and Intelligent Systems, University of Southern California, October 1999.

[63] Paolo Pirjanian and Maja J. Matarić. Multiple objective vs. fuzzy behavior coordination. In *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, volume 61 of *Studies in Fuzziness and Soft Computing*, chapter 10, pages 235–253. Springer-Phisica Verlag, 2001.

[64] Pascal Poupart and Craig Boutilier. VDCBPI: an approximate scalable algorithm for large POMDPs. In *Advances in Neural Information Processing Systems*, pages 1081–1088, Vancouver, BC, 2004.

[65] Doina Precup, Richard S. Sutton, and Satinder P. Singh. Theoretical results on reinforcement learning with temporally abstract options. In *European Converence on Machine Learning*, pages 382–393, 1998.

[66] Jefferson Provost, Benjamin Kuipers, and Risto Miikkulainen. Developing navigation behavior through self-organizing distinctive-state abstraction. *Connection Science*, 18(2):159–172, 2006.

[67] G.V.S. Raju, Jun Zhou, and Roger A. Kisner. Hierarchical fuzzy control. *International Journal of Control*, 54(5):1201–1216, November 1991.

[68] Nelson Ramos, Pedur U. Lima, and J. M. Sousa. Robot behavior coordination based on fuzzy decision-making. In *ROBOTICA 2006 - 6th Portuguese Robotics Festival*, Guimarares, Portugal, 2006.

[69] Michael L. Raymer, William F. Punch, Erik D. Goodman, Leslie A. Kuhn, and Anil K. Jain. Dimensionality reduction using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(2):164–171, 2000.

[70] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, July 1987.

[71] Craig W. Reynolds. Steering behaviors for autonomous characters. In *Proceedings of the Game Developers Conference*, pages 763–782, 1999.

[72] Khashayar Rohanimanesh, Robert Platt Jr., Sridhar Mahadevan, and Roderic A. Grupen. Coarticulation in Markov decision processes. In *Conference on Neural Information Processing Systems*, pages 1137–1144, 2004.

[73] Khashayar Rohanimanesh and Sridhar Mahadevan. Decision-theoretic planning with concurrent temporally extended actions. In *Conference in Uncertainty in Artificial Intelligence*, pages 472–479. Morgan Kaufmann, 2001.

[74] Khashayar Rohanimanesh and Sridhar Mahadevan. Learning to take concurrent actions. In *Conference on Neural Information Processing Systems*, pages 1619–1626. MIT Press, 2002.

[75] Julio K. Rosenblatt. Utility fusion: Map-based planning in a behavior-based system. *Field and Service Robotics*, pages 411–418, 1998.

[76] Julio K. Rosenblatt. Optimal selection of uncertain actions by maximizing expected utility. *Autonomous Robots*, 9(1):17–25, 2000.

[77] Gavin A. Rummery and Mahesan Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR166, Cambridge University, 1994.

[78] Stuart J. Russell and Andrew Zimdars. Q-decomposition for reinforcement learning agents. In *International Conference on Machine Learning*, pages 656–663. AAAI Press, 2003.

[79] Conor Ryan, J. J. Collins, and Michael O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *First European Workshop on Genetic Programming*, volume 1391, pages 83–95, Paris, 1998. Springer-Verlag.

[80] Alessandro Saffiotti. The uses of fuzzy logic in autonomous robot navigation. *Soft Computing*, 1(4):180–197, 1997.

[81] Alessandro Saffiotti and Zbigniew Wasik. Using hierarchical fuzzy behaviors in the robocup domain. In D. Maravall C. Zhou and D. Ruan, editors, *Autonomous Robotic Systems*, pages 235–262. Springer-Verlag, Berlin, DE, 2003.

[82] Satinder P. Singh and David Cohn. How to dynamically merge markov decision processes. In *Advances in Neural Information Processing Systems*, pages 1057–1063. The MIT Press, 1997.

[83] William D. Smart and Leslie Pack Kaelbling. Effective reinforcement learning for mobile robots. In *International Conference on Robotics and Automation*, volume 4, pages 3404–3410. IEEE, 2002.

[84] Nathan Sprague and Dana H. Ballard. Multiple-goal reinforcement learning with modular Sarsa(0). In *International Joint Conference on Artificial Intelligence*, pages 1445–1447, 2003.

[85] Peter Stone and Manuela M. Veloso. Layered learning. In *European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 369–381. Springer, 2000.

[86] Il Hong Suh, Min Jo Kim, Sanghoon Lee, and Byung-Ju Yi. A novel dynamic priority-based action-selection-mechanism integrating a reinforcement learning. In *International Conference on Robotics and Automation*, pages 2639–2646, 2004.

[87] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[88] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[89] Erik Talvitie and Satinder Singh. An experts algorithm for transfer learning. In *International Joint Conference on Artificial Intelligence*, pages 1065–1070, 2007.

[90] Ming Tan. Multi-agent reinforcement learning: Independent versus cooperative agents. In *International Conference on Machine Learning*, pages 330–337, 1993.

[91] Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.

[92] Edward Tunstel. Fuzzy-behavior modulation with threshold activation for autonomous vehicle navigation. In *18th International Conference of the North American Fuzzy Information Procesing Society (NAFIPS)*, pages 776–780, New York, NY, 1999.

[93] Edward Tunstel. Fuzzy-behavior synthesis, coordination, and evolution in an adaptive behavior hierarchy. In *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, volume 61 of *Studies in Fuzziness and Soft Computing*, chapter 9, pages 205–234. Springer-Phisica Verlag, 2001.

[94] Edward Tunstel, Marco A. A. de Oliveira, and Sigal Berman. Fuzzy behavior hierarchies for multi-robot control. *International Journal on Intelligent Systems*, 17(5):449–470, 2002.

[95] Edward Tunstel, Tanya Lippincott, and Mo Jamshidi. Behavior hierarchy for autonomous mobile robots: Fuzzy-behavior modulation and evolution. *International Journal of Intelligent Automation and Soft Computing, Special Issue: Autonomous Control Engineering at NASA ACE Center*, 3(1):37–49, 1997.

[96] Prahlad Vadakkepat, Ooi Chia Miin, Xiao Peng, and Tong Heng Lee. Fuzzy behavior-based control of mobile robots. *IEEE Transactions on Fuzzy Systems*, 12(4):559–565, 2004.

[97] Prahlad Vadakkepat, Xiao Peng, Boon Kiat Quek, and Tong Heng Lee. Evolution of fuzzy behaviors for multi-robotic system. *Robotics and Autonomous Systems*, 55(2):146–161, 2007.

[98] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[99] Steven D. Whitehead. *Reinforcement Learning for the Adaptive Control of Perception and Action*. PhD thesis, University of Rochester, 1992.

[100] Shimon Whiteson, Matthew E. Taylor, and Peter Stone. Empirical studies in action selection with reinforcement learning. *Adaptive Behavior*, 15(1):33, 2007.

[101] Jihoon Yang and Vasant Honavar. Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems and Their Applications*, 13(2):44–49, 1998.

[102] Simon X. Yang, Hao Li, Max Q.H. Meng, and Peter Xiaoping Liu. An embedded fuzzy controller for a behavior-based mobile robot with guaranteed performance. *IEEE Transactions Fuzzy Systems*, 12(4):436–446, 2004.

[103] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

Appendix A

State-Action Spaces

Table A.1: A comparison of the size of the action space is shown for each composite behavior. Also shown is the effect of using different types of behavior hierarchies and the pruning of unacceptable actions. A single hierarchy level represents a monolithic controller comprised of a single behavior which is responsible for accomplishing every primitive task.

Problem Domain	Hierarchy Levels	All Actions		Pruned Actions	
		2D	3d	2D	3d
CA-GS	1	35	175	35	175
	2	25	25	9	9
CA-GS-RA	1	35	175	35	175
	2	125	125	61	61
FLOCKING	1	35	175	35	175
	2	125	125	61	61
FLOCKING-CA	1	35	175	35	175
	2	625	625	369	369
	3	25	25	9	9
FLOCKING-CA-GS	1	35	175	35	175
	2	3,125	3,125	2,101	2,101
	3	25	25	9	9
FLOCKING-CA-GS-RA	1	35	175	35	175
	2	15,625	15,625	11,529	11,529
	3	25	25	9	9

Table A.2: A comparison of the size of the primitive task state spaces is shown for each abstraction level.

Abstraction	2D States	3D States
Full	35	175
Large	25	125
Medium	25	25
Small	5	5

Table A.3: A comparison of the size of the state space is shown for each combination of problem domain and abstracted level.

Problem Domain	Abstraction	2D States	3D States
CA-GS	Full	1,225	30,625
	Large	625	15,625
	Medium	625	625
	Small	25	25
	Minimal	25	25
CA-GS-RA	Full	42,875	5,359,375
	Large	15,625	1,953,125
	Medium	15,625	15,625
	Small	125	125
	Minimal	125	125
FLOCKING	Full	42,875	5,359,375
	Large	15,625	1,953,125
	Medium	15,625	15,625
	Small	125	125
	Minimal	125	125
FLOCKING-CA	Full	1,500,625	937,890,625
	Large	390,625	244,140,625
	Medium	390,625	390,625
	Small	625	625
	Minimal	25	25
FLOCKING-CA-GS	Full	52,521,875	164,130,859,375
	Large	9,765,625	30,517,578,125
	Medium	9,765,625	9,765,625
	Small	3,125	3,125
	Minimal	25	25
FLOCKING-CA-GS-RA	Full	1,838,265,625	28,722,900,390,625
	Large	244,140,625	3,814,697,265,625
	Medium	244,140,625	244,140,625
	Small	15,625	15,625
	Minimal	25	25

Appendix B

Evaluation Environment Parameters

Table B.1: COLLISIONAVOIDANCE environment parameters

Parameter	Value	
	2D	**3D**
Agent initial position	$r = 5$	$r = 5$
Max agent initial velocity	$v = 0.2$	$v = 0.2$
Obstacle count	$[100, 125]$	$[200, 250]$
Obstacle size	$[5, 10]$	$[5, 15]$
Obstacle position	$r = 150$	$r = 125$
Min obstacle-agent distance	8	8
Min obstacle separation	3	2
Max obstacle initial velocity	$v = 0.1$	$v = 0.1$

Table B.2: GOALSEEK environment parameters

Parameter	Value	
	2D	**3D**
Agent initial position	$r = 5$	$r = 5$
Goal distance	$[40, 80]$	$[40, 80]$

Table B.3: RUNAWAY environment parameters

Parameter	Value	
	2D	**3D**
Agent initial position	$r = 5$	$r = 5$
Hazardous object count	$[1, 2]$	$[1, 3]$
Hazardous object strength	$[10, 15]$	$[5, 15]$
Hazardous object position	$r = 25$	$r = 25$
Min hazardous object-agent distance	4	4
Min hazardous object separation	10	10

Table B.4: CA-GS environment parameters

Parameter	Value	
	2D	**3D**
Agent initial position	$r = 5$	$r = 5$
Goal distance	$[30, 60]$	$[30, 60]$
Obstacle count	$[6, 12]$	$[20, 40]$
Obstacle size	$[5, 10]$	$[5, 10]$
Obstacle position	$r = 40$	$r = 40$
Min obstacle-agent distance	8	8
Min obstacle separation	4.5	4.5

Table B.5: CA-GS-RA environment parameters

Parameter	Value	
	2D	3D
Agent initial position	$r = 5$	$r = 5$
Goal distance	$[30, 60]$	$[30, 60]$
Obstacle count	$[6, 12]$	$[20, 40]$
Obstacle size	$[5, 10]$	$[5, 10]$
Obstacle position	$r = 40$	$r = 40$
Min obstacle-agent distance	8	8
Min obstacle separation	4.5	4.5
Hazardous object count	$[2, 4]$	$[4, 8]$
Hazardous object strength	$[1, 5]$	$[1, 5]$
Hazardous object position	$r = 40$	$r = 40$
Min hazardous object-agent distance	4	4
Min hazardous object separation	4	4

Table B.6: FLOCKING environment parameters

Parameter	Value	
	2D	3D
Agent count	$[4, 8]$	$[4, 8]$
Agent initial position	$r = 10$	$r = 10$
Min agent separation	5	5

Table B.7: FLOCKING-CA environment parameters

Parameter	2D Value
Agent count	$[5, 8]$
Agent initial position	$r = 15$
Min agent separation	5
Obstacle count	$[100, 125]$
Obstacle size	$[3, 8]$
Obstacle position	$r = 200$
Min obstacle-agent distance	8
Min obstacle separation	12

Table B.8: FLOCKING-CA-GS environment parameters

Parameter	Value	
	2D	**3D**
Agent count	$[4,6]$	$[4,8]$
Agent initial position	$r = 15$	$r = 15$
Min agent separation	5	5
Goal distance	$[125,150]$	$[125,150]$
Obstacle count	$[50,70]$	$[150,200]$
Obstacle size	$[3,8]$	$[3,8]$
Obstacle position	$r = 150$	$r = 150$
Min obstacle-agent distance	8	8
Min obstacle separation	12	12

Table B.9: FLOCKING-CA-GS-RA environment parameters

Parameter	Value	
	2D	**3D**
Agent count	$[4,6]$	$[4,8]$
Agent initial position	$r = 15$	$r = 15$
Min agent separation	5	5
Goal distance	$[125,150]$	$[125,150]$
Obstacle count	$[60,80]$	$[150,200]$
Obstacle size	$[3,8]$	$[3,8]$
Obstacle position	$r = 150$	$r = 150$
Min obstacle-agent distance	8	8
Min obstacle separation	12	12
Hazardous object count	$[10,14]$	$[20,30]$
Hazardous object strength	$[1,5]$	$[1,5]$
Hazardous object position	$r = 150$	$r = 150$
Min hazardous object-agent distance	4	4
Min hazardous object separation	4	4

Appendix C

Statistical Results

C.1 Student's Paired T-Test Results

To compare the performance of "best-of-run" controllers on the test set, the paired Student's t-test was used. For experiments using reinforcement learning, the policy with the best validation set performance learned at any time throughout the experiment was classified as the "best-of-run." In experiments using grammatical evolution, the "best-of-generation" individual with highest validation set fitness was classified as the "best-of-run." These "best-of-run" controllers where then evaluated on the test set environments. Results presented in **bold** denote results that are statistically significant at at least the 95% confidence level (i.e., $p \leq 0.05$).

Table C.1: Results of the paired Student's t-test for the CA-GS 2D composite task using reinforcement learning with the Bonferroni adjustment are shown. Results presented in **bold** denote results that are statistically significant at at least the 95% confidence level (i.e., $p \leq 0.05$). These results are discussed in Section 7.2.1.

Mean "best-of-run" testing reward				**P-value**
Full	140.7 ± 33.0	Monolithic	148.1 ± 1.1	0.66
		Modular	146.1 ± 11.9	0.91
		Large	140.7 ± 33.0	1.00
		Small	135.0 ± 27.0	0.95
		Modified **Small**	144.5 ± 16.5	0.99
Large	140.7 ± 33.0	Monolithic	148.1 ± 1.1	0.66
		Modular	146.1 ± 11.9	0.91
		Small	135.0 ± 27.0	0.95
		Modified **Small**	144.5 ± 16.5	0.99
Small	135.0 ± 27.0	Monolithic	148.1 ± 1.1	**0.02**
		Modular	146.1 ± 11.9	0.12
		Modified **Small**	144.5 ± 16.5	0.32
Modified **Small**	144.5 ± 16.5	Monolithic	148.1 ± 1.1	1.00
		Modular	146.1 ± 11.9	0.70
Monolithic	148.1 ± 1.1	Modular	146.1 ± 11.9	0.89

Table C.2: Results of the paired Student's t-test for the CA-GS 3D composite task using reinforcement learning with the Bonferroni adjustment are shown. These results are discussed in Section 7.2.1.

Mean "best-of-run" testing reward				**P-value**
Full	146.2 ± 11.9	Monolithic	-23.1 ± 21.4	$\sim$**0.00**
		Modular	-31.0 ± 10.1	$\sim$**0.00**
		Large	148.1 ± 1.2	0.94
		Medium	148.2 ± 0.9	0.92
		Small	148.2 ± 0.8	0.93
		Modified **Small**	146.0 ± 10.3	1.00
Large	148.1 ± 01.2	Monolithic	-23.1 ± 21.4	$\sim$**0.00**
		Modular	-31.0 ± 10.1	$\sim$**0.00**
		Medium	148.2 ± 0.9	1.00
		Small	148.2 ± 0.8	1.00
		Modified **Small**	146.0 ± 10.3	0.80
Medium	148.2 ± 0.9	Monolithic	-23.1 ± 21.4	$\sim$**0.00**
		Modular	-31.0 ± 10.1	$\sim$**0.00**
		Small	148.2 ± 0.8	1.00
		Modified **Small**	146.0 ± 10.3	0.76
Small	148.2 ± 0.8	Monolithic	-23.1 ± 21.4	$\sim$**0.00**
		Modular	-31.0 ± 10.1	$\sim$**0.00**
		Modified **Small**	146.0 ± 10.3	0.76
Modified **Small**	146.0 ± 10.3	Monolithic	-23.1 ± 21.4	$\sim$**0.00**
		Modular	-31.0 ± 10.1	$\sim$**0.00**
Monolithic	-23.1 ± 21.4	Modular	-31.0 ± 10.1	0.24

Table C.3: Results of the paired Student's t-test for the CA-GS-RA 2D composite task using reinforcement learning with the Bonferroni adjustment are shown. These results are discussed in Section 7.2.1.

Mean "best-of-run" testing reward				**P-value**
Full	145.5 ± 7.6	Monolithic	144.5 ± 16.5	1.00
		Modular	144.3 ± 12.0	1.00
		Large	143.5 ± 13.9	0.97
		Small	144.7 ± 8.3	0.84
		Modified **Small**	146.8 ± 0.7	1.00
Large	144.5 ± 16.5	Monolithic	144.5 ± 16.5	1.00
		Modular	144.3 ± 12.0	1.00
		Small	144.7 ± 8.3	0.61
		Modified **Small**	146.8 ± 0.7	1.00
Small	144.7 ± 8.3	Monolithic	144.5 ± 16.5	1.00
		Modular	144.3 ± 12.0	1.00
		Modified **Small**	146.8 ± 0.7	0.52
Modified **Small**	146.8 ± 0.7	Monolithic	144.5 ± 16.5	0.94
		Modular	144.3 ± 12.0	0.73
Monolithic	144.5 ± 16.5	Modular	144.3 ± 12.0	1.00

Table C.4: Results of the paired Student's t-test for the CA-GS-RA 3D composite task using reinforcement learning with the Bonferroni adjustment are shown. These results are discussed in Section 7.2.1.

Mean "best-of-run" testing reward				**P-value**
Full	141.0 ± 25.6	Monolithic	-54.0 ± 4.1	$\sim$**0.00**
		Modular	-57.1 ± 6.3	$\sim$**0.00**
		Large	145.2 ± 8.2	0.93
		Medium	146.8 ± 1.2	0.69
		Small	141.4 ± 20.1	1.00
		Modified **Small**	145.9 ± 7.9	0.87
Large	145.2 ± 8.2	Monolithic	-54.0 ± 4.1	$\sim$**0.00**
		Modular	-57.1 ± 6.3	$\sim$**0.00**
		Medium	146.8 ± 1.2	0.83
		Small	141.4 ± 20.1	0.88
		Modified **Small**	145.9 ± 7.9	1.00
Medium	146.8 ± 1.2	Monolithic	-54.0 ± 4.1	$\sim$**0.00**
		Modular	-57.1 ± 6.3	$\sim$**0.00**
		Small	141.4 ± 20.1	0.49
		Modified **Small**	145.9 ± 7.9	0.98
Small	141.4 ± 20.1	Monolithic	-54.0 ± 4.1	$\sim$**0.00**
		Modular	-57.1 ± 6.3	$\sim$**0.00**
		Modified **Small**	145.9 ± 7.9	0.78
Modified **Small**	145.9 ± 7.9	Monolithic	-54.0 ± 4.1	$\sim$**0.00**
		Modular	-57.1 ± 6.3	$\sim$**0.00**
Monolithic	-54.0 ± 4.1	Modular	-57.1 ± 6.3	0.08

Table C.5: Results of the paired Student's t-test for the CA-GS-RA 2D and 3D composite task using grammatical evolution with the Bonferroni adjustment are shown. These results are discussed in Section 7.2.2.

Mean "best-of-run" testing reward				**P-value**
Full 2D	116.8 ± 53.7	**Full** 3D	137.8 ± 24.2	0.42
		Large 3D	139.4 ± 22.9	0.31
		Medium 3D	139.5 ± 23.0	0.31
		Small 3D	141.7 ± 19.1	0.18
Large 2D	108.8 ± 53.6	**Full** 3D	137.8 ± 24.2	0.09
		Large 3D	139.4 ± 22.9	0.06
		Medium 3D	139.5 ± 23.0	0.06
		Small 3D	141.7 ± 19.1	**0.03**
Small 2D	92.9 ± 59.5	**Full** 3D	137.8 ± 24.2	$\sim$**0.00**
		Large 3D	139.4 ± 22.9	$\sim$**0.00**
		Medium 3D	139.5 ± 23.0	$\sim$**0.00**
		Small 3D	141.7 ± 19.1	$\sim$**0.00**

Table C.6: Results of the paired Student's t-test for the FLOCKING 2D composite task using reinforcement learning with the Bonferroni adjustment are shown. These results are discussed in Section 7.3.1.

Mean "best-of-run" testing reward				**P-value**
Full	55.3 ± 4.2	Monolithic	32.5 ± 4.1	$\sim$**0.00**
		Modular	43.0 ± 0.8	$\sim$**0.00**
		Large	55.3 ± 4.8	1.00
		Small	68.6 ± 2.2	$\sim$**0.00**
Large	55.3 ± 4.8	Monolithic	32.5 ± 4.1	$\sim$**0.00**
		Modular	43.0 ± 0.8	$\sim$**0.00**
		Small	68.6 ± 2.2	$\sim$**0.00**
Small	68.6 ± 2.2	Monolithic	32.5 ± 4.1	$\sim$**0.00**
		Modular	43.0 ± 0.8	$\sim$**0.00**
Monolithic	32.5 ± 4.1	Modular	43.0 ± 0.8	$\sim$**0.00**

Table C.7: Results of the paired Student's t-test for the FLOCKING 2D composite task using grammatical evolution with the Bonferroni adjustment are shown. These results are discussed in Section 7.3.2.

Mean "best-of-run" testing reward				**P-value**
Full	49.7 ± 4.5	Monolithic	25.4 ± 27.7	$\sim$**0.00**
		Large	49.4 ± 9.3	
		Small	43.2 ± 17.9	0.12
Large	49.4 ± 9.3	Monolithic	43.6 ± 5.2	$\sim$**0.00**
		Small	43.2 ± 17.9	0.21
Small	43.2 ± 17.9	Monolithic	43.6 ± 5.2	$\sim$**0.00**

Table C.8: Results of the paired Student's t-test for the FLOCKING-CA 2D composite task using reinforcement learning with the Bonferroni adjustment are shown. These results are discussed in Section 7.3.1.

Mean "best-of-run" testing reward				**P-value**
Full	21.1 ± 3.8	Monolithic	-9.9 ± 7.0	$\sim$**0.00**
		Modular	9.9 ± 1.5	$\sim$**0.00**
		Large	21.7 ± 4.0	0.98
		Small	22.8 ± 4.2	0.31
		Minimal	23.4 ± 4.1	0.07
Large	21.7 ± 4.0	Monolithic	-9.9 ± 7.0	$\sim$**0.00**
		Modular	9.9 ± 1.5	$\sim$**0.00**
		Small	22.8 ± 4.2	0.82
		Minimal	23.4 ± 4.1	0.37
Small	22.8 ± 4.2	Monolithic	-9.9 ± 7.0	$\sim$**0.00**
		Modular	9.9 ± 1.5	$\sim$**0.00**
		Minimal	23.4 ± 4.1	0.99
Minimal	23.4 ± 4.1	Monolithic	-9.9 ± 7.0	$\sim$**0.00**
		Modular	9.9 ± 1.5	$\sim$**0.00**
Monolithic	-9.9 ± 7.0	Modular	9.9 ± 1.5	$\sim$**0.00**

Table C.9: Results of the paired Student's t-test for the FLOCKING-CA 2D composite task using grammatical evolution with the Bonferroni adjustment are shown. These results are discussed in Section 7.3.2.

Mean "best-of-run" testing reward				**P-value**
Full	-1.2 ± 24.9	Monolithic	-89.0 ± 34.6	$\sim$**0.00**
		Large	-0.9 ± 24.1	1.00
		Small	0.5 ± 23.7	1.00
		Minimal	0.7 ± 26.5	1.00
Large	-0.9 ± 24.1	Monolithic	-89.0 ± 34.6	$\sim$**0.00**
		Small	0.5 ± 23.7	1.00
		Minimal	0.7 ± 26.5	1.00
Small	0.5 ± 23.7	Monolithic	-89.0 ± 34.6	$\sim$**0.00**
		Minimal	0.7 ± 26.5	1.00
Minimal	0.7 ± 26.5	Monolithic	-89.0 ± 34.6	$\sim$**0.00**

Table C.10: Results of the paired Student's t-test for the FLOCKING-CA-GS 2D composite task using reinforcement learning with the Bonferroni adjustment are shown. These results are discussed in Section 7.3.1.

Mean "best-of-run" testing reward				**P-value**
Full	123.0 ± 1.1	Monolithic	-100.7 ± 56.2	$\sim$**0.00**
		Modular	121.4 ± 4.2	0.17
		Full Original Alg.	120.3 ± 2.6	0.00
		Large	122.8 ± 1.6	1.00
		Small	122.1 ± 1.9	0.10
		Minimal	121.9 ± 1.5	$\sim$**0.00**
		Modified **Small**	121.3 ± 1.3	$\sim$**0.00**
Large	122.8 ± 1.6	Monolithic	-100.7 ± 56.2	$\sim$**0.00**
		Modular	121.4 ± 4.2	0.35
		Small	122.1 ± 1.9	0.53
		Minimal	121.9 ± 1.5	0.08
		Modified **Small**	121.3 ± 1.3	$\sim$**0.00**
Small	122.1 ± 1.9	Monolithic	-100.7 ± 56.2	$\sim$**0.00**
		Modular	121.4 ± 4.2	0.95
		Minimal	121.9 ± 1.5	1.00
		Modified **Small**	121.3 ± 1.3	0.29
Minimal	121.9 ± 1.5	Monolithic	-100.7 ± 56.2	$\sim$**0.00**
		Modular	121.4 ± 4.2	1.00
		Modified **Small**	121.3 ± 1.3	0.63
Modified **Small**		Monolithic	-100.7 ± 56.2	$\sim$**0.00**
		Modular	121.4 ± 4.2	1.00
Monolithic	-100.7 ± 56.2	Modular	121.4 ± 4.2	$\sim$**0.00**

Table C.11: Results of the paired Student's t-test for the FLOCKING-CA-GS 2D composite task using grammatical evolution with the Bonferroni adjustment are shown. These results are discussed in Section 7.3.2.

Mean "best-of-run" testing reward				**P-value**
Full	121.4 ± 2.8	Monolithic	-114.0 ± 57.9	$\sim$**0.00**
		Large	121.7 ± 2.6	1.00
		Small	121.7 ± 3.0	1.00
		Minimal	117.3 ± 17.3	0.78
Large	121.7 ± 2.6	Monolithic	-114.0 ± 57.9	$\sim$**0.00**
		Small	121.7 ± 3.0	1.00
		Minimal	117.3 ± 17.3	0.71
Small	121.7 ± 3.0	Monolithic	-114.0 ± 57.9	$\sim$**0.00**
		Minimal	117.3 ± 17.3	0.73
Minimal	117.3 ± 17.3	Monolithic	-114.0 ± 57.9	$\sim$**0.00**

Table C.12: Results of the paired Student's t-test for the FLOCKING-CA-GS-RA 2D composite task using reinforcement learning with the Bonferroni adjustment are shown. These results are discussed in Section 7.3.1.

Mean "best-of-run" testing reward				**P-value**
Full	118.0 ± 2.3	**Large**	117.0 ± 2.9	0.33
		Small	115.5 ± 3.6	$\sim$**0.00**
		Minimal	117.2 ± 2.7	0.58
		Modified **Small**	116.9 ± 2.8	0.28
Large	117.0 ± 2.9	**Small**	115.5 ± 3.6	0.26
		Minimal	117.2 ± 2.7	1.00
		Modified **Small**	116.9 ± 2.8	1.00
Small	115.5 ± 3.6	**Minimal**	117.2 ± 2.7	0.99
		Modified **Small**	116.9 ± 2.8	0.25
Minimal	117.2 ± 2.7	Modified **Small**	116.9 ± 2.8	0.99

C.2 Randomized Two-Way ANOVA Results

To compare both the rates at which controllers were developed and their resulting performance, the randomized two-way ANOVA test was used [60]. The tables below describe the results of these tests for the following effects:

A Effect Performance of the agent
B Effect The number of updates to the Q-values

Results denoted by **X** denote results that are statistically significant at the 95% confidence level (i.e., $p \leq 0.05$).

Table C.13: Results of the randomized two-way ANOVA for the CA-GS 2D composite task using reinforcement learning are shown. **X** denotes results that are statistically significant at the 95% confidence level. These results are discussed in Section 7.2.1.

		A Effect	B Effect	Interaction Effect
Full	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Large	**X**	**X**	
	Small	**X**	**X**	**X**
	Modified **Small**	**X**	**X**	**X**
Large	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Small	**X**	**X**	**X**
	Modified **Small**			
Small	Monolithic		**X**	**X**
	Modular	**X**	**X**	**X**
	Modified **Small**	**X**	**X**	**X**
Modified **Small**	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
Monolithic	Modular	**X**	**X**	**X**

Table C.14: Results of the randomized two-way ANOVA for the CA-GS 3D composite task using reinforcement learning are shown. These results are discussed in Section 7.2.1.

		A Effect	B Effect	Interaction Effect
Full	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Large	**X**	**X**	
	Medium	**X**	**X**	
	Small	**X**	**X**	**X**
	Modified **Small**	**X**	**X**	
Large	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Medium	**X**		
	Small	**X**	**X**	**X**
	Modified **Small**			
Medium	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Small	**X**	**X**	**X**
	Modified **Small**			
Small	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Modified **Small**	**X**	**X**	**X**
Modified **Small**	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
Monolithic	Modular			

Table C.15: Results of the randomized two-way ANOVA for the CA-GS-RA 2D composite task using reinforcement learning are shown. These results are discussed in Section 7.2.1.

		A Effect	B Effect	Interaction Effect
Full	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Large			
	Small	**X**	**X**	**X**
	Modified **Small**	**X**	**X**	
Large	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Small	**X**	**X**	**X**
	Modified **Small**	**X**	**X**	**X**
Small	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Modified **Small**	**X**	**X**	**X**
Modified **Small**	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
Monolithic	Modular	**X**	**X**	**X**

Table C.16: Results of the randomized two-way ANOVA for the CA-GS-RA 3D composite task using reinforcement learning are shown. These results are discussed in Section 7.2.1.

		A Effect	B Effect	Interaction Effect
Full	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Large	**X**	**X**	
	Medium	**X**	**X**	**X**
	Small	**X**	**X**	**X**
	Modified **Small**	**X**	**X**	**X**
Large	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Medium	**X**	**X**	**X**
	Small		**X**	**X**
	Modified **Small**	**X**	**X**	**X**
Medium	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Small	**X**	**X**	**X**
	Modified **Small**		**X**	**X**
Small	Monolithic			
	Modular	**X**	**X**	
	Modified **Small**	**X**	**X**	**X**
Modified **Small**	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
Monolithic	Modular			**X**

Table C.17: Results of the randomized two-way ANOVA for the CA-GS-RA 2D and 3D composite task using grammatical evolution are shown. These results are discussed in Section 7.2.2.

		A Effect	B Effect	Interaction Effect
Full 2D	**Full** 3D	**X**	**X**	
	Large 3D	**X**	**X**	
	Medium 3D	**X**	**X**	
	Small 3D	**X**	**X**	
Large 2D	**Full** 3D	**X**	**X**	
	Large 3D	**X**	**X**	
	Medium 3D	**X**	**X**	
	Small 3D	**X**	**X**	
Small 2D	**Full** 3D	**X**	**X**	
	Large 3D	**X**	**X**	
	Medium 3D	**X**	**X**	
	Small 3D	**X**	**X**	

Table C.18: Results of the randomized two-way ANOVA for the FLOCKING 2D composite task using reinforcement learning are shown. These results are discussed in Section 7.3.1.

		A Effect	B Effect	Interaction Effect
Full	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Large			
	Small		**X**	
Large	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Small			
Small	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
Monolithic	Modular		**X**	

Table C.19: Results of the randomized two-way ANOVA for the FLOCKING-CA 2D composite task using reinforcement learning are shown. These results are discussed in Section 7.3.1.

		A Effect	B Effect	Interaction Effect
Full	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Large			
	Small			
	Minimal			
Large	Monolithic	**X**	**X**	
	Modular	**X**	**X**	**X**
	Small			
	Minimal			
Small	Monolithic	**X**	**X**	
	Modular	**X**	**X**	**X**
	Minimal			
Minimal	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
Monolithic	Modular	**X**	**X**	

Table C.20: Results of the randomized two-way ANOVA for the FLOCKING-CA-GS 2D composite task using reinforcement learning are shown. These results are discussed in Section 7.3.1.

		A Effect	B Effect	Interaction Effect
Full	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Full Original Alg.	**X**	**X**	**X**
	Large	**X**		
	Small	**X**	**X**	**X**
	Minimal	**X**	**X**	
	Modified **Small**	**X**	**X**	
Large	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Small	**X**	**X**	**X**
	Minimal	**X**	**X**	
	Modified **Small**	**X**	**X**	**X**
Small	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Minimal	**X**	**X**	**X**
	Modified **Small**	**X**	**X**	**X**
Minimal	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
	Modified **Small**	**X**	**X**	**X**
Modified **Small**	Monolithic	**X**	**X**	**X**
	Modular	**X**	**X**	**X**
Monolithic	Modular	**X**	**X**	**X**

Table C.21: Results of the randomized two-way ANOVA for the FLOCKING-CA-GS-RA 2D composite task using reinforcement learning are shown. These results are discussed in Section 7.3.1.

		A Effect	B Effect	Interaction Effect
Full	**Large**	**X**	**X**	**X**
	Small	**X**	**X**	
	Minimal	**X**	**X**	
	Modified **Small**			
Large	**Small**	**X**	**X**	**X**
	Minimal	**X**	**X**	**X**
	Modified **Small**	**X**	**X**	
Small	**Minimal**	**X**	**X**	**X**
	Modified **Small**	**X**	**X**	**X**
Minimal	Modified **Small**	**X**	**X**	**X**

Appendix D

Sample Policies

Table D.1 details an example of an effective policy learned for the COLLISIONAVOIDANCE primitive task.

Table D.2 details an example of an effective policy learned for the CA-GS composite task using the **Small** abstraction level. Note that in the vast majority of stats, the COLLISIONAVOIDANCE primitive task is given a `MEDIUM` DOA, or weight, while the GOALSEEK primitive task is given a `FULL` DOA.

Table D.3 details an example of an effective policy learned for the FLOCKING-CA composite task using the **Minimal** abstraction level. Note that in all but two states, the FLOCKING composite task is given a `LOW` DOA, or weight, while the COLLISIONAVOIDANCE primitive task is given a `FULL` DOA. This indicates that the learned policy is risk-averse and tends to avoid collisions at all costs. Note that the COLLISIONAVOIDANCE task is able to avoid collisions with both obstacles and other agents. As a result, even though the SEPARATION primitive task modulated by the FLOCKING composite task may receive a `LOW` DOA, the COLLISIONAVOIDANCE task is still able to avoid agent-agent collisions.

Table D.1: An effective policy for the COLLISIONAVOIDANCE primitive task learned by composite reinforcement learning

State		Action	
Collision Dir	**Time till collision**	**Steer Yaw**	**Steer Speed**
BACK_LEFT	NOW	RIGHT	FASTER
BACK_LEFT	REAL_SOON	RIGHT	MUCH_FASTER
BACK_LEFT	SOON	RIGHT	SAME
BACK_LEFT	LONG_TIME	RIGHT	SAME
BACK_LEFT	DISTANT	CENTER	SAME
LEFT	NOW	RIGHT	FASTER
LEFT	REAL_SOON	SMALL_RIGHT	SLOWER
LEFT	SOON	RIGHT	SAME
LEFT	LONG_TIME	RIGHT	SAME
LEFT	DISTANT	CENTER	SAME
SMALL_LEFT	NOW	RIGHT	FASTER
SMALL_LEFT	REAL_SOON	SMALL_LEFT	MUCH_SLOWER
SMALL_LEFT	SOON	RIGHT	SAME
SMALL_LEFT	LONG_TIME	RIGHT	SAME
SMALL_LEFT	DISTANT	RIGHT	SAME
CENTER	NOW	RIGHT	FASTER
CENTER	REAL_SOON	SMALL_LEFT	SAME
CENTER	SOON	RIGHT	SAME
CENTER	LONG_TIME	RIGHT	SAME
CENTER	DISTANT	RIGHT	SAME
SMALL_RIGHT	NOW	SMALL_LEFT	MUCH_FASTER
SMALL_RIGHT	REAL_SOON	SMALL_LEFT	SAME
SMALL_RIGHT	SOON	SMALL_LEFT	SAME
SMALL_RIGHT	LONG_TIME	SMALL_LEFT	SAME
SMALL_RIGHT	DISTANT	CENTER	SAME
RIGHT	NOW	SMALL_LEFT	MUCH_FASTER
RIGHT	REAL_SOON	SMALL_LEFT	SAME
RIGHT	SOON	SMALL_LEFT	SAME
RIGHT	LONG_TIME	CENTER	SAME
RIGHT	DISTANT	CENTER	SAME
BACK_RIGHT	NOW	SMALL_LEFT	MUCH_FASTER
BACK_RIGHT	REAL_SOON	SMALL_LEFT	SAME
BACK_RIGHT	SOON	CENTER	SAME
BACK_RIGHT	LONG_TIME	CENTER	SAME
BACK_RIGHT	DISTANT	CENTER	SAME

Table D.2: An effective policy for the CA-GS composite task using the **Small** abstraction level learned by composite reinforcement learning

State		**Action**	
CA **priority**	GS **priority**	CA **DOA**	GS **DOA**
ZERO	ZERO	MEDIUM	FULL
ZERO	LOW	MEDIUM	FULL
ZERO	MEDIUM	MEDIUM	FULL
ZERO	HIGH	MEDIUM	FULL
ZERO	FULL	MEDIUM	FULL
LOW	ZERO	MEDIUM	FULL
LOW	LOW	MEDIUM	FULL
LOW	MEDIUM	MEDIUM	FULL
LOW	HIGH	MEDIUM	FULL
LOW	FULL	MEDIUM	FULL
MEDIUM	ZERO	MEDIUM	FULL
MEDIUM	LOW	MEDIUM	FULL
MEDIUM	MEDIUM	MEDIUM	FULL
MEDIUM	HIGH	MEDIUM	FULL
MEDIUM	FULL	FULL	MEDIUM
HIGH	ZERO	MEDIUM	FULL
HIGH	LOW	MEDIUM	FULL
HIGH	MEDIUM	MEDIUM	FULL
HIGH	HIGH	MEDIUM	FULL
HIGH	FULL	FULL	MEDIUM
FULL	ZERO	MEDIUM	FULL
FULL	LOW	MEDIUM	FULL
FULL	MEDIUM	FULL	HIGH
FULL	HIGH	MEDIUM	FULL
FULL	FULL	FULL	MEDIUM

Table D.3: An effective policy for the FLOCKING-CA composite task using the **Minimal** abstraction level learned by composite reinforcement learning

State		**Action**	
FLOCKING **priority**	CA **priority**	FLOCKING **DOA**	CA **DOA**
ZERO	ZERO	LOW	FULL
ZERO	LOW	LOW	FULL
ZERO	MEDIUM	LOW	FULL
ZERO	HIGH	LOW	FULL
ZERO	FULL	LOW	FULL
LOW	ZERO	LOW	FULL
LOW	LOW	LOW	FULL
LOW	MEDIUM	LOW	FULL
LOW	HIGH	LOW	FULL
LOW	FULL	LOW	FULL
MEDIUM	ZERO	LOW	FULL
MEDIUM	LOW	LOW	FULL
MEDIUM	MEDIUM	LOW	FULL
MEDIUM	HIGH	LOW	FULL
MEDIUM	FULL	LOW	FULL
HIGH	ZERO	LOW	FULL
HIGH	LOW	LOW	FULL
HIGH	MEDIUM	LOW	FULL
HIGH	HIGH	LOW	FULL
HIGH	FULL	LOW	FULL
FULL	ZERO	LOW	FULL
FULL	LOW	FULL	MEDIUM
FULL	MEDIUM	FULL	MEDIUM
FULL	HIGH	LOW	FULL
FULL	FULL	LOW	FULL

Appendix E

Fuzzy Membership Functions

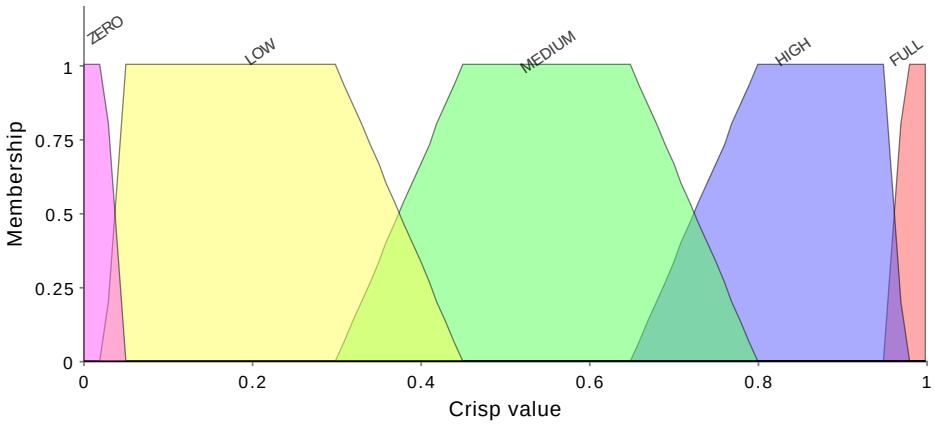


Figure E.1: Behavior DOA linguistic variable

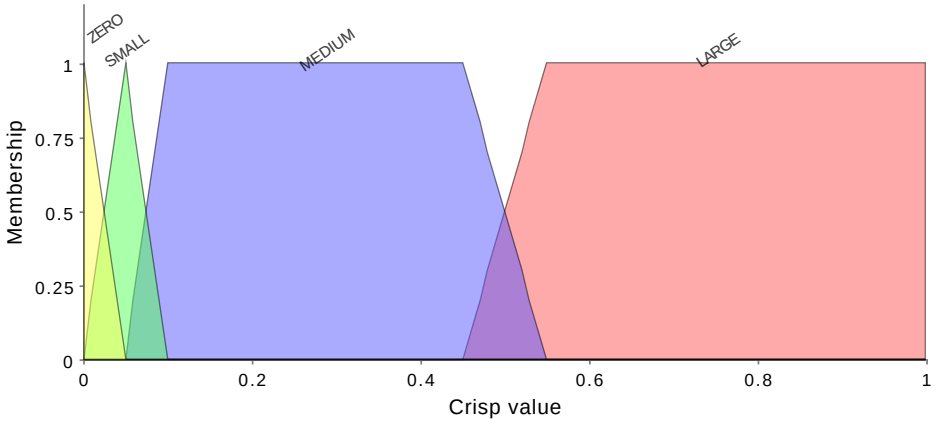


Figure E.2: Direction delta linguistic variable

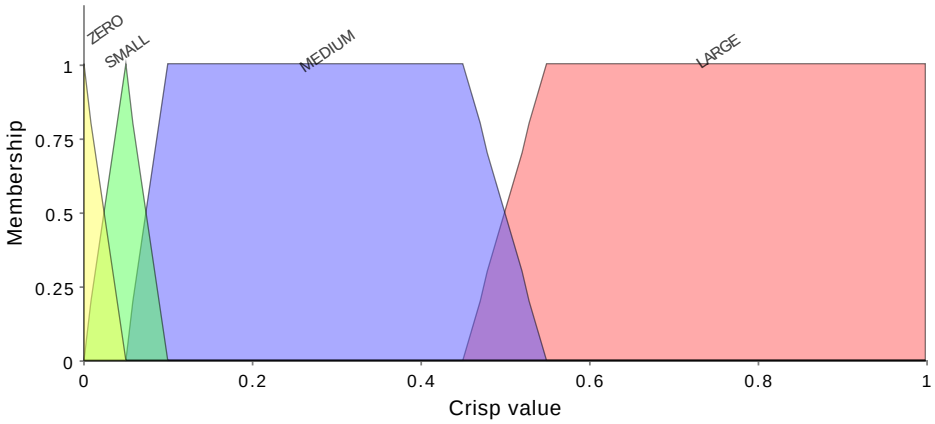


Figure E.3: Phi direction delta linguistic variable

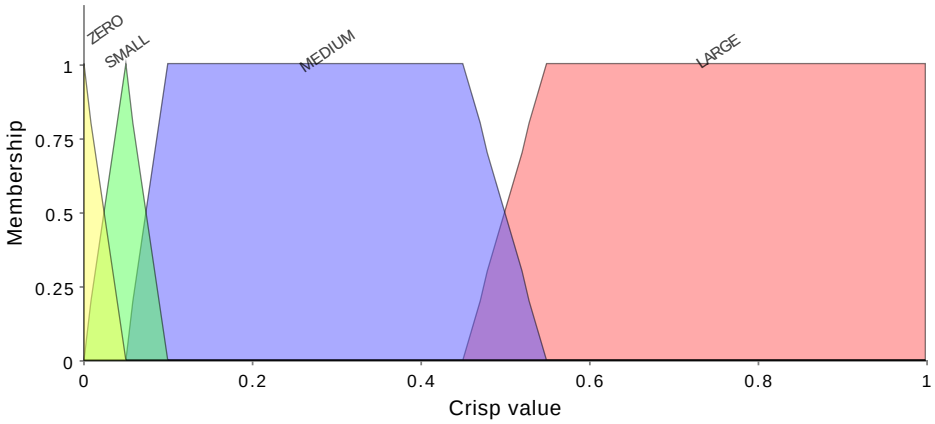


Figure E.4: Phi direction error delta linguistic variable

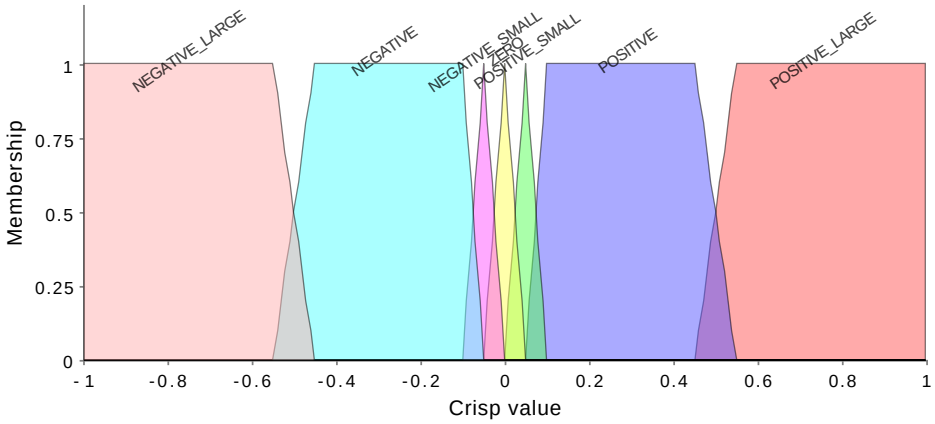


Figure E.5: Phi direction error linguistic variable

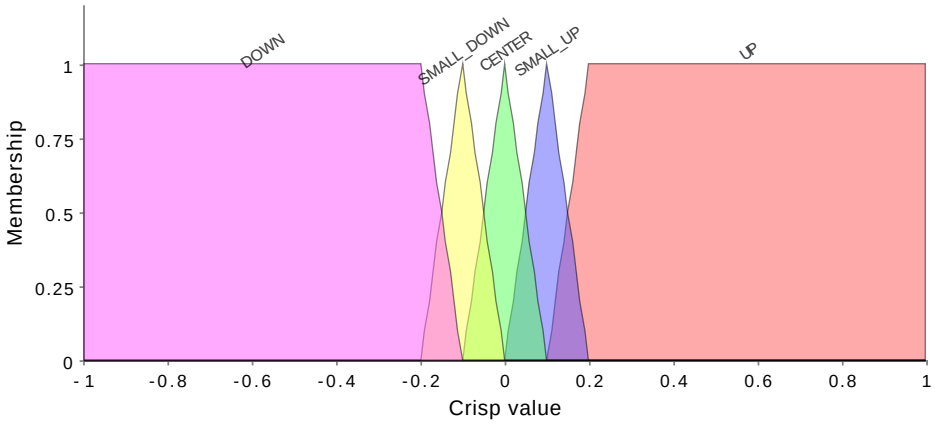


Figure E.6: Phi direction linguistic variable

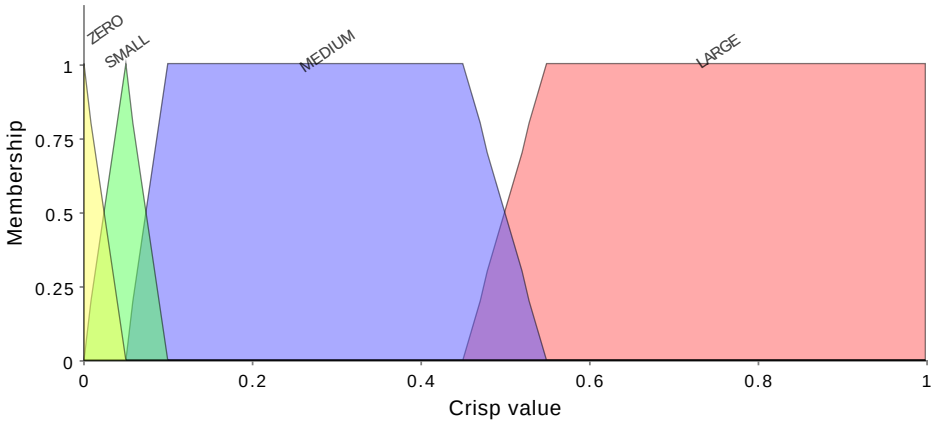


Figure E.7: Theta direction delta linguistic variable

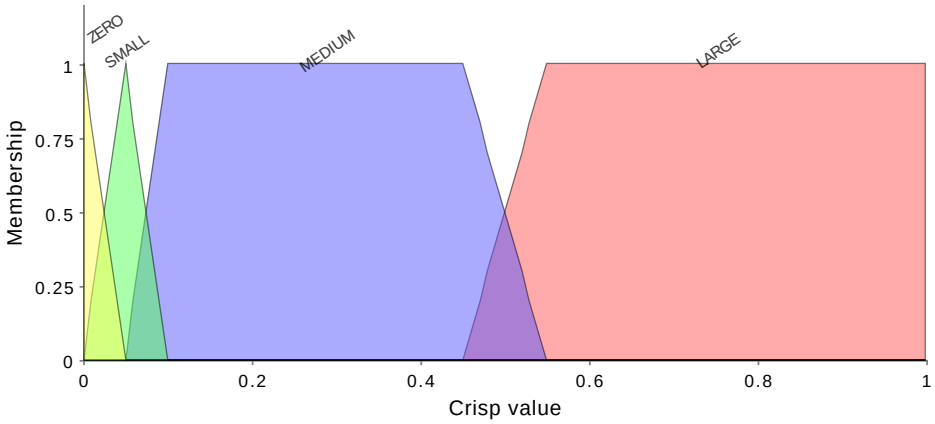


Figure E.8: Theta direction delta error linguistic variable

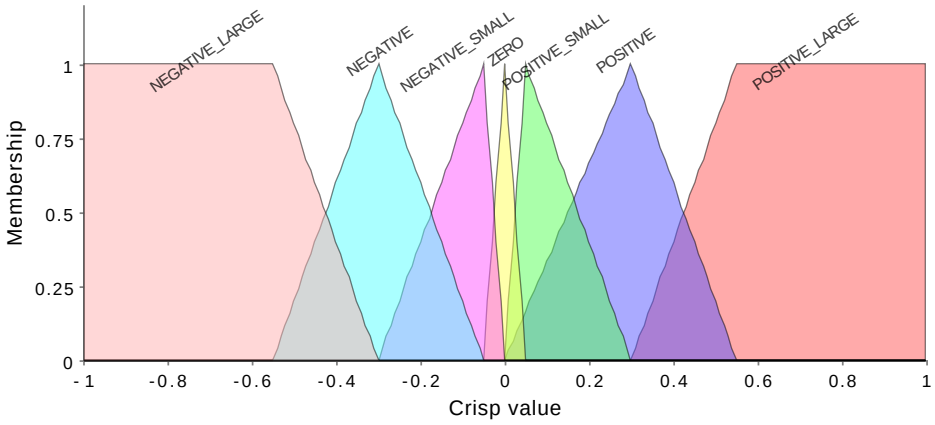


Figure E.9: Theta direction error linguistic variable

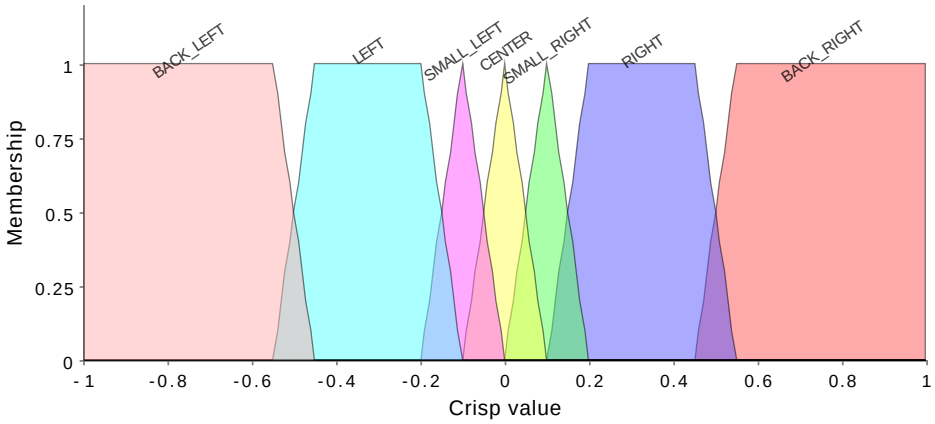


Figure E.10: Theta direction linguistic variable

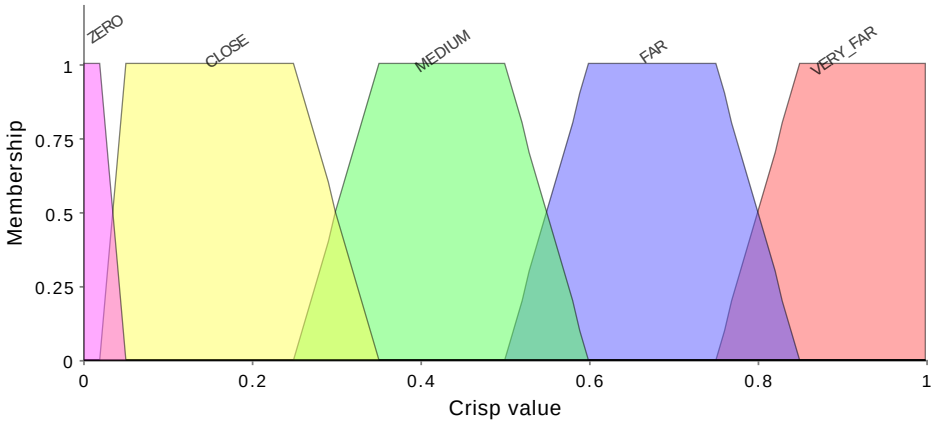


Figure E.11: Distance linguistic variable

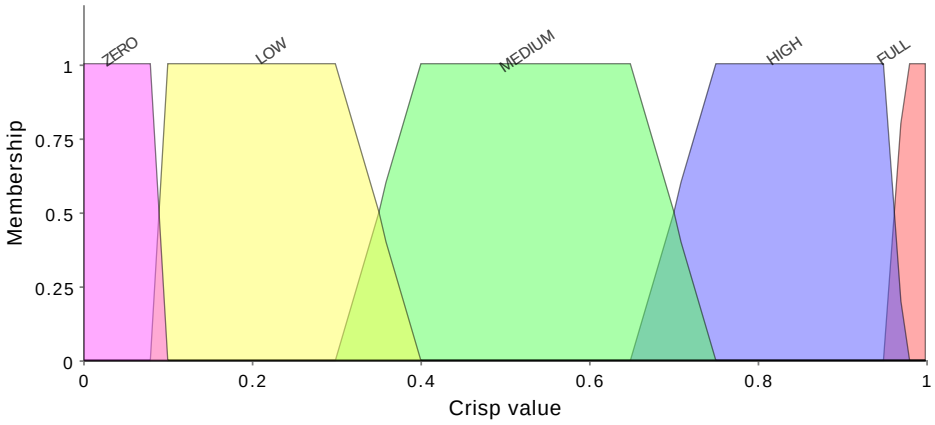


Figure E.12: Priority linguistic variable

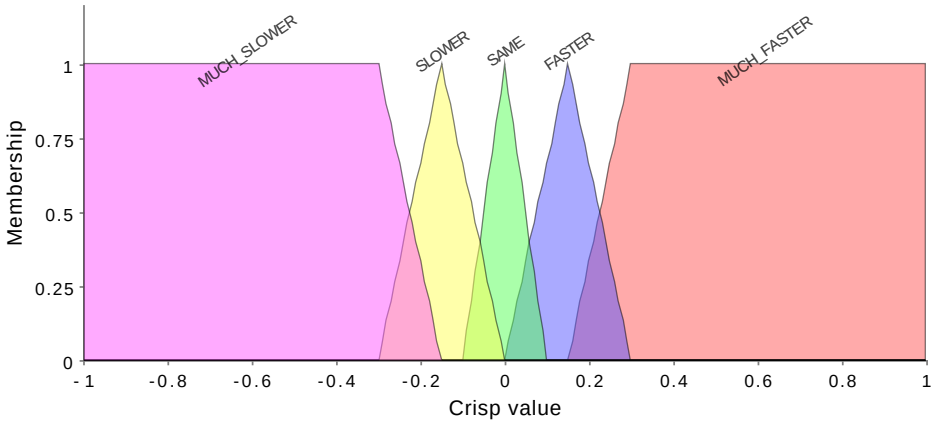


Figure E.13: Speed difference linguistic variable

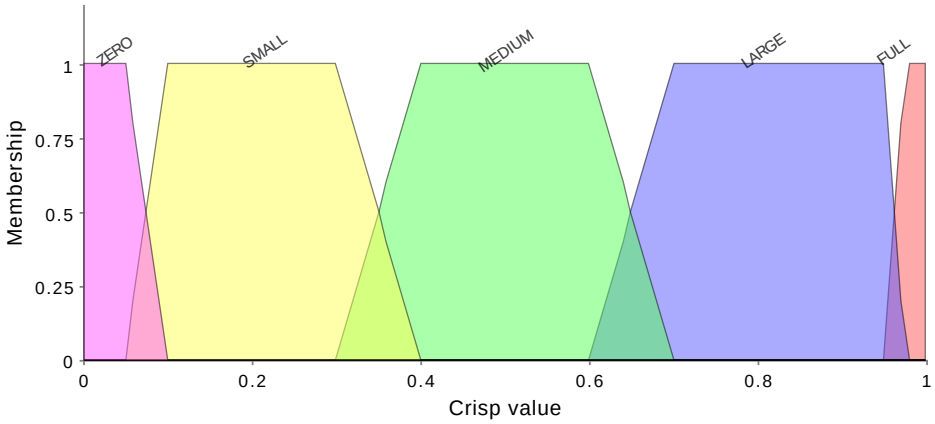


Figure E.14: Strength linguistic variable

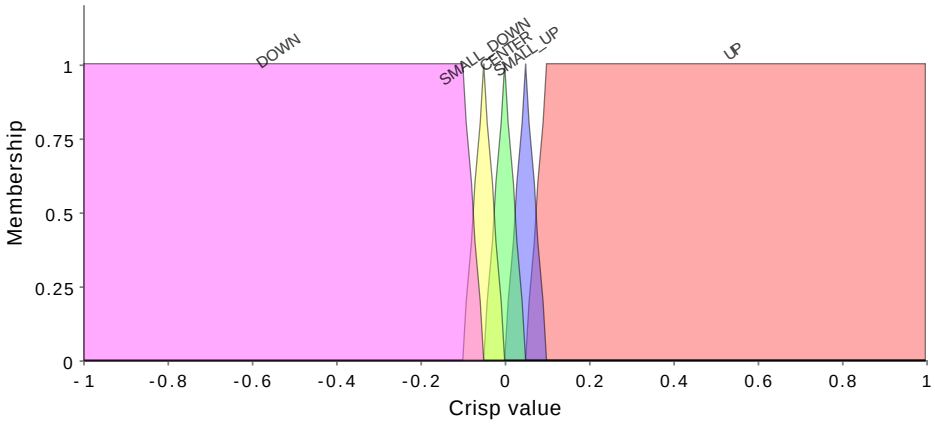


Figure E.15: Steering pitch linguistic variable

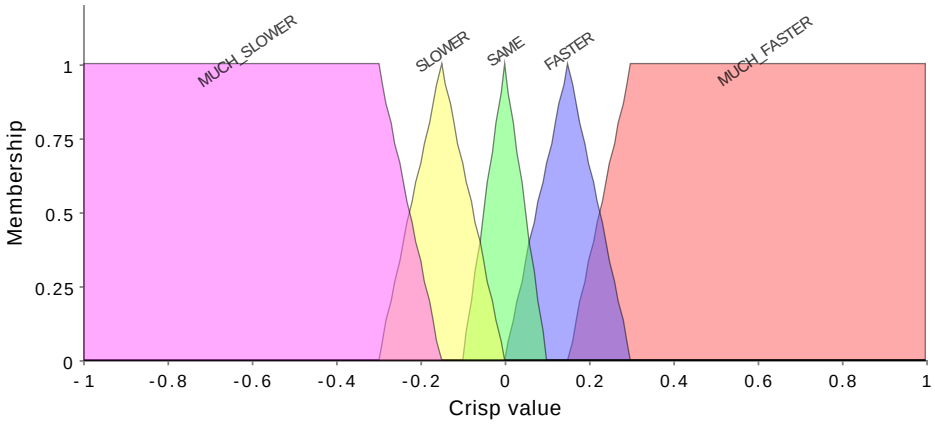


Figure E.16: Steering speed linguistic variable

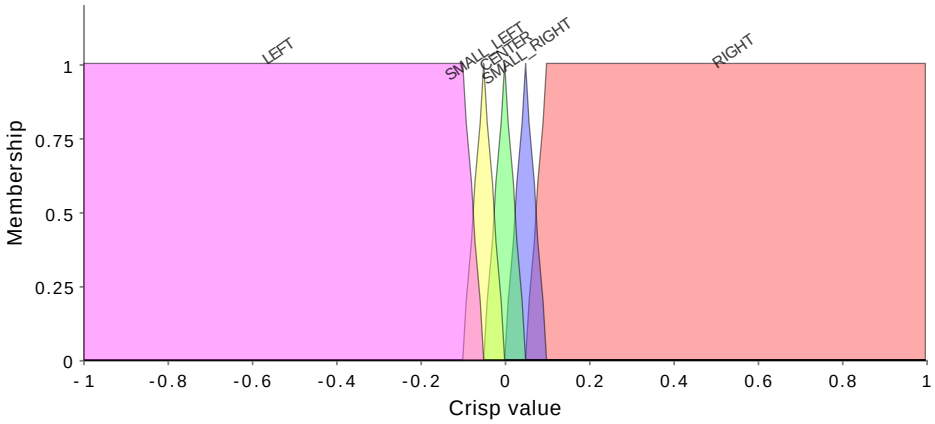


Figure E.17: Steering yaw linguistic variable