growing up virtual:
the computational lessons of development

by

Derek Eugen Lyons
B.A. Chemistry, Reed College, 2000
M.Sc. Engineering Science, University of Oxford, 2002

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2004

author .... [illegible] ....
Program in Media Arts and Sciences
May 7, 2004

certified by .... [illegible] ....
Bruce M. Blumberg
Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Thesis Supervisor

accepted by ....
Andrew B. Lippman
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

growing up virtual:

the computational lessons of development

by

Derek Eugen Lyons

Submitted to the Program in
Media Arts and Sciences,
School of Architecture and Planning,
on May 7, 2004, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Arts and Sciences

abstract

Cognitive development is one of nature's most important mechanisms for creating robustly adaptive intelligent creatures. From felids to oscines, developing animals are capable of learning in adverse environments with a reliability that often outpaces the current state-of-the-art in artificial intelligence (AI). The purpose of this thesis, therefore, is to examine how insights from cognitive development might be applied to the design of AI architectures. Starting with a targeted review of the ethological literature, I identify the key computational lessons of development, the fundamental conceptual insights that suggest intriguing new strategies for behavioral organization. These insights are then employed in the design of a developmental behavior architecture in which a hierarchical motivation-based behavior system is coupled to a distributed set of domain-specific learning tools. The architecture is deployed in a synthetic character (Hektor the mouse) whose challenge is to learn to play a competitive card matching game successfully against a human user. Evaluation of Hektor's performance on this task, at both qualitative and quantitative levels of description, reveal that the developmental architecture is capable of surmounting complex learning objectives in a novel and efficient manner. I conclude that the architecture presented here represents a valuable starting point for further consideration of developmental design principles.

Thesis Supervisor: Bruce M. Blumberg
Title: Associate Professor of Media Arts and Sciences

4|

growing up virtual:

the computational lessons of development

by

Derek Eugen Lyons

The following people served as readers for this thesis:

thesis reader [illegible]
Cynthia Breazeal, Sc.D.
Assistant Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Massachusetts Institute of Technology

thesis reader [illegible]
Raymond Coppinger, Ph.D.
Professor of Biology
Hampshire College

acknowledgments

A great many people have helped the ideas in this thesis grow up during my time at the Media Lab.

First and foremost I would like to sincerely thank my advisor Bruce Blumberg. Bruce's work has been an inspiration to me since I first read about Silas more than seven years ago, and it has been a privilege to develop my own research under his guidance. I am particularly indebted to Bruce for encouraging me to freely explore my interests in development, and for giving shape and refinement to so many of my core ideas with his own insights.

I would also like to thank the other members of the Synthetic Characters Group, both past and present, for their friendship and intellectual community. Bill Tomlinson and Marc Downie were the sage elders who helped me get me feet wet working on *AlphaWolf*, and who provided me with the daunting examples of creativity and engineering elegance that I continue to strive for in my own work. Special thanks to Bill for showing me that you can be both a new assistant professor and a wicked sharp dresser at the same time. Marc has made an everlasting imprint on my life by introducing me to optical margin alignment, increasing my level of gray scale discrimination, and showing me the beauty of lower case headings. I dedicate the low percentage of capital letters in this thesis to Marc's influence.

Daphna Buchsbaum suffered through the trauma that was the butterfly with me, and has since engaged in a noble campaign against my various OCD tendencies. My future office mates all owe Daphna, and her drinking birds, a significant debt of gratitude. Matt Berlin has been my partner in total mind and body control; it has been a pleasure to go through both stressful sponsor meetings and belt exams with him – and of course to exchange the occasional jumping side kick to the solar plexus. Jennie Cochran has been a great companion on the character's morning shift over the past two years, and an unfailing provider of dirty looks when I propose working too hard or too long for my own good. Thanks also to Jennie for the yellow volleyball. Jesse Gray is the nicest leet haxor that I've ever met. Even though he never did make it to class with Matt and I, I still hold out hope of seeing his yellow belt skills one day. Aileen Kawabe is a paragon of patience and organization in the midst of ever encroaching chaos, and has made my time here run much more smoothly in innumerable ways. My sincere thanks to Aileen for administrative support, ergonomic advice, and friendship.

In addition to the characters I'd also like to thank the other members of the 1CC posse that have made my time here memorable, in particular Ari Benbasat (for his steadfast sarcasm) and Josh Lifton (for not telling me what I was getting into when he took me to my first Tae Kwon Do class).

I'd like to especially acknowledge the efforts of my readers, Cynthia Breazeal and Ray Coppinger. Cynthia's work on Kismet has been a great inspiration to me in my journey towards developmental creatures. I thank Ray for sharing his amazing insight into animal cognitive development; all of my own ideas on the subject begin with his writings. Both have helped me to significantly sharpen the presentation of this thesis, and the blame for any remaining errors or infelicities rests solely with me.

Many thanks to Linda Peterson and Pat Solakoff for making MAS HQ a friendly place, and for helping me to deal with the kinds of seemingly insurmountable administrative obstacles that can make strong men weep. The fact that this thesis has been completed on time owes more than a little to their patient shepherding and support.

Finally, as in every good thing that I do, I want to thank my family for the love and support that made it possible in the first place. Mom, Ron, Brooke, and Rusty: you are the foundation of my every success.

table of contents

list of figures

introduction 1

1.1 on the perils of intuition

"The challenge is to forget about how we think the world ought to work, and instead figure out how it works in fact."
(Searle, 1998)

This thesis concerns itself with the matter of cognitive development, a topic with which we all have a considerable degree of first-hand experience. When it comes to thinking closely about the processes and results of development, therefore, it is remarkably easy to feel that we have a firm intuitive grasp of the phenomenon. Development, our intuitive logic tells us, is the process of transitioning from a cognitively and physically helpless infant form to an independent adult form. Development is about 'growing up,' or progressing in a smooth and continuous way from the primitive to the sophisticated.

From this perspective, the idea of applying the concept of development to the design of artificially intelligent robots or virtual creatures may appear to be a somewhat esoteric intellectual enterprise. Why, one might ask, should we make a computational system that has to go through the bother of growing up anyway? Do we really want to burden ourselves with moody adolescent robots? What, other than a clever piece of engineering, would a computational model of development represent? I argue that the more refined understanding of development possessed by ethologists and psychologists holds the answers to all of these questions. The goal of this thesis is to begin making connections to this rich body of existing knowledge, demonstrating how insights gained from the study of cognitive development in living systems can open exciting new possibilities for AI.

1.2 thesis organization

The conceptual groundwork for this thesis begins in **chapter 2: a computational perspective on development**, in which I argue that the intuitive notion of development, despite its seeming obviousness, is actually deeply misleading in important ways. In particular, I use knowledge drawn from the ethology and psychology research communities to debunk what I refer to as the intuitive myth of development, the misunderstanding amongst many computer scientists and artificial intelligence researchers that ontogeny is a process of linearly increasing organismic complexity. Beginning from the intellectual perspective of Coppinger [23], I show that development is better conceptualized as a progression through differentially specialized forms, each of which is sophisticated in the context of its unique ecological niche.

Taking this refined understanding of the phenomenon as a starting point, I then articulate the position that development holds fundamental insights into the question of how to best design robust and adaptive artificial creatures. A careful analysis of the ethological literature is used to synthesize the **key computational lessons of development**, lessons that I argue form the conceptual basis for a compelling new approach to AI. A review of related work in the AI and machine learning communities in **chapter 3: relationship to prior computational work** positions this new approach with respect to its computational antecedents.

In **chapter 4: a developmental behavior architecture**, I describe how the lessons learned from the biological literature can be applied to the implementation of a new style of computational behavior system. Following theories of hierarchical behavioral organization articulated by investigators such as Baerends, Blumberg, Dawkins, Hogan, Timberlake, and Tinbergen ([1], [6], [24], [25], [70], [34], [69], [35], [71]), I define a behavior architecture in which a creature's behavioral capabilities are organized into a motivation-based hierarchy. Comprised of modular **behavioral elements** arranged in a reconfigurable lattice structure and coordinated by top-level **motivational subsystems**, the architecture supports the kind of evolving specialization that is a hallmark of development in living systems [23]. Learning is integrated into the architecture in the form of **specialized learning tools** (SLTs), distributed domain-specific learning mechanisms inspired by the cognitive specializations observed to guide learning in the ontogeny of many animal species. I argue that arranging SLTs in cooperative hierarchies is a sensible and flexible means of achieving scalable behavioral adaptation.

Having defined the general framework of the developmental behavior architecture, **chapter 5: hektor: a computational case study** describes how the architecture is applied to a specific problem of computational learning. **Hektor** is a mouse who is motivated by his ever-present desire for cheese, which can be won by wagering with the user on a competitive card matching game called **CheeseWhiz**. Hektor's learning task is to derive the user-determined matching rules that govern the game in order to amass a glorious fortune in cheese. What might otherwise be a trivial problem of exhaustive search, however, is complicated by the fact that the user periodically has the ability to secretly alter the rules of the game; Hektor must thus deploy an intelligent and efficient learning strategy in order to be successful. The architecture of the developmental behavior system is further elaborated and clarified by describing how it maps onto Hektor's particular learning challenge. The organization of Hektor's behavior system is detailed, as well as the set of four independent SLTs that combine to drive his learning.

Hektor is put to the test in **chapter 6: evaluation and discussion**, which describes a series of experiments used to evaluate his proficiency at the

CheeseWhiz task. Detailed qualitative description of Hektor's behavior and learning are presented in order to help the reader appreciate how the diverse mechanisms of the developmental architecture are combined and deployed in practice. A series of quantitative studies then undertake to formally evaluate Hektor's learning performance and compare it to that which would be expected from a naive search strategy. It is found that Hektor's developmental architecture allows him to learn much more efficiently than might otherwise be possible, enabling him to compete effectively with the human user even under very adverse experimental conditions.

Finally, in **chapter 7: conclusions and future directions**, I summarize the contributions of this thesis, and make some suggestions for future research. I conclude that this thesis provides a valuable proof-of-concept for the applicability of developmental design principles to AI, and argue that there is a wealth of possibilities for future exploration.

Now, without further ado, let's commence with the real work of this thesis and move on to consider the ground truth of cognitive development as it occurs in living systems.

a computational perspective on development 2

In this chapter I undertake a targeted review of the ethological literature in order to motivate the computational importance of development. The central question of "why development?" is answered by adopting the perspective on the phenomenon advocated by the noted canine ethologist Ray Coppinger [23]. In this conception, development is not about simple things becoming more complex; it is rather about managing complexity in response to the demands of the creature's ever-changing environment. Development, in other words, is largely about organizing, supervising, and integrating the results of learning. Two key lessons bearing on this point are elucidated from the literature and used to make suggestions as to how developmental design motifs might benefit AI architecture; these design motifs will serve as the guiding principles for the developmental architecture described in **chapter 4**. Finally, the developmental approach to AI is put in context by considering its relationship to prior computational work.

2.1 why development?

Let us begin by directly confronting the primary question that is raised by this work: why development? That is, if our ultimate goal is to create an adult virtual creature, why should we do so by way of the putatively primitive infant and juvenile forms?

The answer to this dilemma comes from recognizing, as ethologists and psychologists would, that the question itself rests on a false premise. The view that a creature's early ontogenic forms are simply incomplete approximations to its final adult state is a manifestation of the intuitive myth of development. According to this misconception, widespread amongst AI and computer science researchers, development is a process of linearly increasing complexity wherein the creature goes from a primitive infant state to a well-adapted adult state. While this sort of description may be a reasonably apt one for the morphological changes that occur during ontogeny, it is not at all accurate with respect to the accompanying cognitive and behavioral changes.

Coppinger has written with great effectiveness on the myth of linear development [23]. He remarks:

> Ontogenic growth is not linear progression; rather, the maturing animal undergoes complex redefinition at distinct stages, often adding or subtracting whole organ systems, necessitating a concomitant reintegration of the whole organism at each stage.

Contrary to intuition, the process of development is one of discontinuous change and large-scale behavioral reorganization; it is a process of transitioning between distinct specialized forms. As we shall see, replacement and restructuring, not just refinement, are critical to the processes of development.

It is very important to recognize that the specialized forms through which a creature transitions during ontogeny are not orderable on some grand scale of overall complexity; rather, each is uniquely adapted to face the demands that will confront the creature during the corresponding period in development. In considering the mammalian neonate, for example, we should be careful not to let the form's physical helplessness fool us into regarding it as fundamentally primitive. Rather, the crucial thing to focus on is the fact that the neonate is supremely well-adapted to its ontogenic niche. Natural selection does not begin once a creature reaches adulthood – it operates at all stages in development. Both the infant and the adult, therefore, represent "the most efficient and competitive uses of form to capture energy in their respective niches" [23]; they are simply different specialized forms for achieving different specialized ends.

So, returning to our starting point, why development? We have seen that development, properly understood, is a process that is centrally concerned with adapting creatures in a flexible and robust way to dynamically changing environments. The computational significance of development in this light is quite clear. If we can create artificial creatures which are as nimbly adaptive to changing circumstances as developing animals, then we will have come a very long way indeed from the current state-of-the-art in artificial intelligence. In the rest of this chapter, I will demonstrate how this degree of sure-footedness in a changing environment is in fact a direct result of the infant to juvenile to adult movement of ontogeny.

2.2 development and learning

At this point in the discussion we must confront one of the most nettlesome theoretical issues facing any developmental research program: an articulation of the difference between development and learning. In particular, now that I have argued that development is fundamentally a process of adapting to a changing environment, I must ward off the assertion that development is simply learning carried out over a longer time scale.

Ethologists and psychologists have expended a considerable amount of effort arguing over the distinction between learning and development (reviewed in [1] and [41]). Opinions have spanned the full range of possibilities, from denying that development is anything other than learning to arguing that traditional learning processes provide virtually no insight into development [1]. From a more dispassionate perspective, it seems unlikely that either of these extreme positions is correct. As Shettleworth has argued, both learning and development are processes by which experience shapes the behavior of an organism [63]; this commonality makes it unsurprising that, in many respects, learning and development should be deeply intertwined.

In this chapter I will present a version of this more modern 'interactionist' synthesis (see, for example, [22], [45], [41], [31], [65], [66], [32]) which holds that development and learning are separable but tightly coupled processes. More specifically, I will argue that development is the phenomenon that guides learning so as to make it both feasible and adaptive. The adaptive component of this argument is particularly important as it runs counter to the seemingly sensible proposition that learning is always a good thing. In fact, the effortful process of learning should be engaged only when it is necessary, e.g. when an organism is imperfectly adapted to its environment [23]. The most fit creature, in other words, is not necessarily the one that learns the most, rather it is the one that learns the right things at the right times in order to take optimal advantage of its niche.

Development is the process that helps the creature direct and constrain its learning in this adaptive fashion.

In the coming sections of this chapter, I will argue that the key conceptual lessons of development all bear on the fashion in which development interacts with learning. Specifically, I will show that one of the most powerful aspects of development is the manner in which it allows an organism to learn in the appropriate context, thereby greatly simplifying the task. Similarly, I will argue that another critical feature of development is its use of specialized tools for learning, effectively deploying prior knowledge of the learning task to help make it more tractable.

2.3 a brief word on mechanism

While the definition of a specific computational architecture for development will be considered in detail in **chapter 4**, it is helpful to our current purposes have a high-level intuition in mind regarding the mechanistic underpinnings of the phenomenon. To provide this intuition, let us consider the general style of cognitive architecture which developmentalists attribute to animal minds, and consider what insights this suggests for the basic attributes of a computational model.

A computationally well-defined view of development begins with the behavior systems tradition, as exemplified by the work of authors such as Baerends, Blumberg, Dawkins, Hogan, Timberlake, and Tinbergen ([1],

[6], [24], [25], [70], [34], [69], [35], [71]). From our perspective, the important aspect of the behavior systems view is its highly modular representation of behavioral organization. In broad terms, a behavior system can be conceptualized as consisting of an organized collection of specialized behavioral modules, each of which is defined in terms of the simple function which it executes. While the precise definition of these modules is not important for the purposes of this discussion, we can usefully follow Hogan for an illustrative example [34]. Hogan expresses his behavior systems in terms of central, perceptual, and motor modules. Central modules are conceptualized as corresponding to particular functional drives such as feeding or reproduction; they help to coordinate the activity of the perceptual and motor modules to which they are attached. In a cow, for example, the central module for feeding might orchestrate the activity of a perceptual module responsible for locating inviting patches of grass, and a motor module that controls the grazing motor pattern.

The importance of this kind of modular organization arises from its dynamic flexibility. Critically, a modular cognitive architecture suggests a coherent way of conceptualizing the discontinuous changes that occur during development. In particular, modular organization provides a convenient (if admittedly abstract) mechanistic interpretation for the prior statement that the operations of replacement and reorganization are fundamental during ontogeny; replacement and reorganization can be thought of as referring to 're-wiring' of the connections between the modules in the behavior system. This kind of large-scale reorganization amongst the modules, in addition to refinement taking place within the modules themselves, is fundamental to development. In Hogan's words, "The study of development is primarily the study of changes in [modules] and in the connections among them" [34]. Throughout the course of this chapter, I will rely on the notion of re-wiring connections amongst the elements of a modular behavior system to supply a high-level mechanistic interpretation for developmental change.

Having now set the stage with a more accurate understanding of development, let us move on to the second half of this chapter, an investigation of the phenomenon's key computational lessons.

2.4 a unifying theme: constrain and conquer

We are now in a position to begin exploring the 'big lessons' of development – the central insights that make the phenomenon so useful as a means for thinking about the design of intelligent artificial creatures. Before doing this, however, it is useful to consider the unifying theme that positions these lessons in relationship to one another, a theme that I term **constrain and conquer.**

As we will soon see, each of the big lessons of development is fundamentally concerned with managing the learning process in a clever and flexible way. The consensus that emerges from these lessons is that attempting to create a monolithic, domain-general learning algorithm is not the best means of achieving flexible learning. Rather, robust and adaptive learning comes from the imposition of meaningful constraints to guide and simplify the process. Development, in this view, can be understood as nature's means of providing these constraints.

I will present two broad classes of developmental constraints for close analysis. First, it will be shown that the process of development helps creatures to learn the right things in the right contexts. The critical insight here, simple to articulate but powerful in practice, is that the optimal context for learning a given skill will often be quite different from the context in which that skill will ultimately be expressed. Secondly, I will present examples of how development simplifies many learning tasks by endowing creatures with highly specialized, domain-specific learning tools. I will argue that development illustrates how a plurality of special-purpose learning mechanisms can often achieve far superior results to a single general mechanism.

2.5 lesson one: learn in the right contexts

Research in artificial intelligence has traditionally been focused on the how of learning: what is the algorithmic machinery that must be in place for particular kinds of learning to occur? When one looks to development for inspiration, however, it quickly becomes clear that in nature, it is often the *when* of learning that is the dominant question. The key insight that evolution seems to have hit on is that by situating learning in the right context, it is often possible to greatly simplify the problem.

In this section, we will consider the implications of this first big lesson of development. The subject of neonatal feeding behavior will be discussed as a source of biological inspiration, one that gives rise to several key points in a preliminary computational definition of the 'right' context for learning. We will then go on to consider several corollaries to this central lesson, drawing insight from the subjects of instinctive behavior and juvenile play to suggest additional design guidelines for a developing artificial creature.

"A surprising fact about the feeding behavior of many neonatal animals is that their early feeding movements are relatively independent of motivational factors associated with food deprivation."
(Hogan, 1994)

2.5.1 biological inspiration: neonatal feeding behavior

As the sidebar quote intimates, the feeding behavior of neonates is a surprisingly counterintuitive phenomenon. In particular, it has been shown by numerous independent investigators that the feeding of baby rats, puppies, kittens, and even humans is largely independent of nutritional state ([33], [30]). Why should this be so?

Hogan, through his work on the early feeding behavior of chicks, has presented an intriguing framework for understanding this developmental puzzle [34]. Hogan begins by observing that while chicks first display pecking (the key motor pattern involved in their feeding behavior) within hours of birth, this pecking is initially independent of nutritional state. Early pecking appears to be better described as exploratory in nature,[1] and may be directed towards a variety of food and non-food targets in the environment. The connection between pecking and the cessation of hunger does not begin to form until several days after the chick has hatched, and even then it must be mediated by specific functional experience; the chick must experience pecking followed by ingestion to form the critical association. The chick, in other words, must *learn* that pecking is a reliable strategy for reducing hunger rather than being given this knowledge innately.

On the surface, all of Hogan's findings are deeply counterintuitive; they seem to fly in the face of any standard notion regarding the design of a complex computational system. The design begs the question: why leave critical parameters of a system, parameters whose values are known a priori, unspecified? Why isn't the all-important association between pecking and eating simply hard-wired into the chick?

In order to answer this question we must admit one more important fact about chicks. Specifically, it turns out that for the first few days of life, chicks simply don't need to eat; their nutritional requirements are met

1. Hogan, in fact, likens this early pecking to play. This comparison is actually quite suggestive, as we shall discover in the discussion on the role of play behavior in mammalian neonates.

entirely by the residual effects of the egg's yolk. The implication of this fact is that were pecking initially associated with the reduction of hunger, it would be highly unlikely to be expressed to any significant degree during the first few days of life; the motivation for its expression would simply be absent. This hypothetical situation would be deeply problematic for the chick's development, insofar as early exploratory pecking appears to be critical for refining the perceptual mechanisms that the chick uses to identify food. As Hogan notes, "The putative food-recognition mechanism in newly hatched chicks must be largely unspecified because of the very wide range of stimuli that are characteristic of items that chicks will come to accept as food" [34]; it is the process of early exploratory pecking that serves the critical function of optimizing this food-recognition apparatus for the chick's immediate environment. Hence, a chick that was not motivated to peck for the first few days of its life, which indeed did not peck at all until it first experienced hunger, would find itself in a very dangerous situation. Such a chick would be faced with the prospect of learning a non-trivial amount of information (the refined motor pattern and its appropriate eliciting stimulus in the environment) quickly and accurately, or starving for want of success.

The lesson in all of this is clear: the optimal context for learning a particular behavior is often not the context in which the behavior will ultimately be expressed. Why wait to learn how to obtain food, in other words, until you're already hungry? Learning is best conducted before it is critical, such that errors can be corrected rather than incurring disastrous consequences. Miller articulates this idea very nicely, remarking on what he terms "the anticipatory nature of development" [51]:

> [T]he capacity to exhibit species typical behavior develops prior to the time that an organism actually needs to exhibit such behavior in adapting to its ecological niche.

Though this point can seem rather obvious in retrospect, it must be noted that it has not often been explicitly recognized in the design of artificial learning systems. It is often somehow easier or more intuitive to think of learning as occurring in the same context as that of the desired final behavior. In the next section, I will attempt to begin redressing this deficiency by defining some of the key features that might be used to identify advantageous learning contexts in a computational setting.

2.5.2 computational implications: what defines the right context for learning?

As the designers of AI systems, how can we go about identifying the right learning contexts for our creatures? While specific policies must naturally be tailored to specific situations, I would argue that the preceding discussion has illuminated several general design guidelines.

the right context makes learning safe

This is the design guideline that resonates most strongly in the example provided by Hogan's chicks. The right context for learning is one that is safe, that will not strongly punish the creature for mistakes or failures. As has already been noted, one direct consequence of this guideline is that learning contexts and expression contexts will very often be distinct. Whenever it is possible to learn a skill or a skill component in a situation other than that in which the skill is demanded, doing so will almost always be to the creature's advantage.

It should be noted that the ability to learn in safe contexts in the fashion being described makes several interesting demands on an artificial creature's design. First, the creature must have some ability to flexibly define the relevant reward signals for guiding its learning. To help unpack this statement, note that when a behavior is learned in the context in which it will be expressed, the relevant reward signal is often inherent in the behavior's function. For example, if a chick were to learn how to peck in the context of being hungry, then the reward signal specifying a successful peck would be the accompanying ingestion of food. However, as we have seen, chicks initially learn how to peck in an exploratory context, not one motivated by their nutritive state. In the exploratory context, what is the reward signal that the chick uses to identify examples of well-formed pecks? It is as though the chick must have some innate representation for what a good peck feels like proprioceptively, and is able to use this knowledge to generate an implicit reward signal for refining its motor behavior. To generalize, the salient point is that the chick is able to use subtle, problem-specific reward signals to guide its learning in contexts where the learned behavior is not yet functional. This is likely to be a very important point in the design of developmental AI systems, and it is one to which we will return to later in this chapter.

The second design demand that learning in safe contexts imposes is that the products of learning must be capable of being repurposed into new domains; they must be redeployable from the context in which they were initially acquired to the context in which they will ultimately be most useful. Fortunately, the modular cognitive architecture that we have previously discussed provides a convenient mechanism for this kind of knowledge redeployment. That is, at an abstract level, the activity of learning and expressing a behavior in different contexts can be understood in terms of *re-wiring* the connections between modules in the behavior system. The changing motivational context for pecking behavior in chicks, for example, can be explained in terms of the pecking motor module forming a new connection to the central mechanism corresponding to hunger.[2] The modularity of the cognitive building blocks, in other words, makes it con-

2. Of course, just because the observed behavior can be conceptually explained in this manner does not mean that this process (taken at a suitable level of abstraction) is actually the full mechanistic story in the chick. In addition to (or rather than) being motivated by hunger, for example, the pecking motor module may well be self-rewarding in the same manner that many components of felid predatory behavior are thought to be [43]. The point being made here is simply that a modular cognitive architecture allows us to tell a *plausible* story regarding knowledge redeployment, if not one that is necessarily isomorphic with biological reality.

ceptually simple for the products of learning to be reused in dynamic ways.

the right context separates distinct learning objectives

A close consideration of Hogan's experiments reveals a second design guideline: the right context should make it possible to separate distinct learning objectives. To see what is meant by this, note that developing chicks actually have to learn several distinct things about their pecking behavior; they must learn not only what pecking is useful for, but also how to peck effectively. One of the important consequences of the fact that pecking is initially exploratory rather than functional in nature is that the chick is able to approach these two learning problems separately. The chick learns how to peck before it has pressing nutritional demands, and then learns why to peck after it has mastered the relevant motor program.

The critical point here is that by selecting learning contexts intelligently, it is often possible to tease apart the distinct elements of a compound learning problem. Once again, this design guideline reaffirms the importance of keeping the learning context distinct from the expression context. If a behavior can be learned before it needs to be functional, then the learning process can often be made more tractable by dividing it into its component parts.

the right context makes learning easier

This is something of a meta-guideline, as it is implicit in all of the points that have been mentioned previously. However, it is worth mentioning explicitly here in order to emphasize the fact that it is often much simpler to learn behaviors in unexpected contexts. Take for example the phenomenon of birdsong learning, which we will consider in much greater detail in **lesson two**. The evidence suggests that the first phase of birdsong learning occurs soon after chicks hatch, when they form an auditory template for their species-typical song on the basis of the vocalization of conspecifics. Why should this template be formed so early in ontogeny, often months before the chick itself ever needs to vocalize? The answer is that this is the time in the chick's life in which it is easiest to have continuous access to the songs of conspecifics; for a newly hatched chick sitting in the nest, the vocalizations of its parents are likely to be among the most distinct and noticeable of the stimuli available in the environment. The lesson is that by freeing creatures of the constraint that learning and expression must occur in the same context, learning can migrate to the contexts that help to make it easiest.

Having now introduced the first big lesson of development, let us turn our attention to some of its important corollaries. In the following sections, I will expand my consideration of advantageous learning contexts by considering the role that multiple contexts and opportunistic contexts seem to play in development.

2.6 first corollary to lesson one: multiple contexts for learning

The first corollary to **lesson one** regards the notion of multiple learning contexts. The inspiration for this point derives from the development of instinctive (e.g. highly stereotyped and species-specific) forms of behavior. As we will see, the biological evidence suggests that the stability of instinctive behavior within species is maintained not just by learning in the right context, but by having multiple right contexts to learn in. This finding has several interesting implications for our artificial creatures, regarding both the importance of multiple learning contexts and the philosophical question of learning versus hard-wiring.

2.6.1 biological inspiration: instinctive behavior

The ethologist David Miller has pointed out that many researchers have made the mistake of regarding the 'development of instinctive behavior' as something of an oxymoron [51]. After all, isn't instinctive behavior simply genetically programmed? While genetic programs may be involved at some level, the fact remains that genes code for proteins, not for behaviors. In other words, "claiming that a behavior is caused by genes ... does not bring us any closer to understanding the behavior's development than to claim that it is caused by ecology, neurobiology, anatomy and so forth"

[51]. Instinctive behavior is therefore as legitimate a target for developmental investigation as any other kind of behavior [40]. Indeed, by understanding how instinctive behavior typically arises during ontogeny, we can understand something very important about how nature goes about 'programming' its most robust forms of behavior.

Miller advances an intriguing hypothesis regarding the development of instinctive behavior. Rather than regarding instinctive behavior as somehow privileged, he argues that it generally arises from experiential factors in a fashion that is analogous to the development of 'ordinary' behavior. The only difference is that instinctive behaviors generally arise in less obvious ways from more subtle, and often multiple, forms of experience; "the greater the stereotypy [e.g. instinctiveness] of the behavioral outcome, the less obvious are the experiences that influence its development." On this account, instinctive behaviors are simply those whose emergence is highly canalized in the sense of Waddington [73]; their development is supported by multiple independent experiential pathways (in addition to relevant genetic and environmental factors), the collective effect of which is to make the emergence of the behavioral in question a virtual certainty.

As an example in support of Miller's hypothesis, let us consider the mallard duckling. Since mallards are ground-nesting birds, mothers whose ducklings have just hatched must be particularly vigilant for predators lurking near the nesting site. If the mother detects a threat in the vicinity of the nest, she will issue an alarm call as a warning to her brood. Mallard ducklings respond instinctively to this call by immediately ceasing all vocal and locomotor activity. The response is so highly stereotyped that large broods of 8 to 12 ducklings can be completely 'frozen' for multiple minutes at a time by a single exposure to the call.

So how does this instinctive behavior pattern come to be so reliably encoded in the duckling, to the extent that it is ready to be perfectly expressed virtually from the moment of hatching? In order to answer this question, Miller undertook a variety of careful experiments designed to isolate the specific experiential factors that appeared causal in the development of the freezing response. The first finding was that ducklings required some experience of their own perinatal vocalizations, or those of their littermates, in order to be reliably receptive to the alarm call; ducklings which were incubated and hatched under devocalized, socially isolated conditions did not manifest the freezing behavior. Note that it is not direct (e.g. functional) experience of the alarm call itself that the ducklings require to develop the freezing response, simply general experience of vocalization.

This finding in itself may not be so surprising. One can imagine that the perinatal experience of vocalization is allowing the ducklings to parameterize some behaviorally important auditory template with general information about their conspecifics' vocal patterns, an idea that we will return to in a slightly different context during **lesson two**. The truly remarkable finding was the discovery that the freezing behavior of ducklings deprived of auditory experience could be repaired through purely social experience. That is, while devocalized ducklings raised in isolation had little sensitivity to the alarm call, devocalized ducklings raised in groups responded in a fashion that was virtually indistinguishable from that of normal ducklings! Note that these socially raised ducklings still had no experience with conspecific vocalization; their receptiveness to the alarm call was somehow being repaired through an alternate experiential pathway. Though the precise mechanism of this social remediation is not yet clear, this remarkable result does demonstrate the importance that multiple learning contexts play in development. As Miller puts it [51]:

> Metaphorically, just as a person can travel from one city to another by means of different routes, so are there multiple developmental trajectories connecting experiential events and behavioral outcomes. A block in one path can be circumvented if development is allowed to detour to another path.

2.6.2 computational implications: multiple learning strategies versus innateness

I would argue that Miller's ducklings, and the subject of instinctive behavior generally, have two important computational implications. First, they argue that learning can be made significantly more robust simply by supporting it with multiple learning contexts. As implied by the duckling example, the learning of particularly important skills should be approached in such a way that an unexpected difficulty in one learning context can be compensated for by the availability of another, independent context. Second, the idea of multiple learning contexts is sufficiently powerful that it can, in many cases, be trusted to generate behavior that we might otherwise be tempted to build-in. Just as multiple learning contexts can make instinctive behavior highly reliable without it being strongly innate, so can they generate critical computational behaviors in the place of rigid hard-wiring.

But why is this important? What's wrong, after all, with simply hard-wiring a behavior that we known in advance will be necessary? The most obvious answer to these questions is that even behaviors that can be largely specified in advance will often have subtle parameters that cannot be completely predicted a priori; there will almost always be some aspect of the behavior that can be better specified at run-time, in reference to the creature's specific, unpredictable environment, than it can be from the proverbial drawing board. By allowing such behaviors to be learned within a robust combination of independent learning contexts, a greater degree of adaptedness and flexibility can be achieved. A second and more subtle answer has to do with the benefits that can be derived from the dynamic reorganization of a behavior system, as occurs for example during learning. In the next section, I will argue that this process of reorganization may carry unanticipated advantages in the form of opportunistic learning.

2.7 second corollary to lesson one: opportunistic contexts for learning

Not all potentially advantageous learning contexts can be predicted in advance. How can we go about designing artificial creatures that can discover, and make profitable use of, opportunistic contexts for learning? In this section, I will argue that the developmental phenomenon of play may hold the answer to this question. In particular, play demonstrates how the metamorphic restructuring of a behavior system, as occurs during the juvenile phase of ontogeny, provides exactly the right circumstance for a high-degree of exploratory, opportunistic learning.

2.7.1 biological inspiration: play

The playfulness of childhood is the most demanding teacher we have.
(Minsky, 2002)

The inspiration for this section comes from that quintessential behavioral trait of juvenile mammals: play. While it is easy to dismiss the phenomenon of play as non-functional, a simple manifestation of the exuberance of youth, some ethologists such as Coppinger have suggested that play may actually be serving a deeply important purpose.

Let us begin by considering the essential difficulty inherent in the juvenile stage of ontogeny. Coppinger frames this difficulty by observing that while the infant and the adult both have well-defined ecological niches, the juvenile stage of ontogeny is one that, by definition, is not quite adapted to either. Indeed, the juvenile never quite attains an adaptive fit to its environment, existing instead in a constant state of cognitive and morphological flux.[3] Coppinger and Smith put the point rather poetically [23]:

> [The juvenile] has to metamorphose from a dependent form to an independent form. It has to rebuild the morphology from the neonate's niche to the adult's niche. It has to survive as the neonatal behavior wanes and the adult behavior waxes. In other words, it has to survive as the stereotyped neonatal form and behavior is pulled apart and the adult form and behavior is assembled. *It has to live in the house as it is being rebuilt.* (emphasis mine)

The analogy to remodeling a home seems quite instructive here. The juvenile period can be understood as one of profound remodeling in the behavior system, with the architecture that defined the neonatal mind being gradually re-assembled and re-wired into the adult architecture. The real question then is how any organism survives the slings and arrows of adolescence. How can a creature survive a lengthy phase in ontogeny in which it is never fully adapted to its environment?

The answer to this conundrum may be derived by considering how the apparent disadvantages of the juvenile can be turned to its advantage. To wit, because of the metamorphic state of its behavior system, the juvenile has a remarkable amount of freedom to experiment with applying the various components of its behavioral repertoire in novel situations. Rather

3. A phenomenon that many of us regrettably experienced first-hand in junior high school.

than being constrained by a "closed-form" behavior system in which motor patterns are tightly organized into stereotyped sequences subserving specific functional goals (e.g. hunting, reproducing, grooming, etc.), the juvenile can engage more atomic components of its behavior in more variable contexts. Leyhausen provides an excellent illustration of this phenomenon in his writings on developmental processes in juvenile cats [43]. He observes that the elements of the adult hunting sequence (for example, lying in wait, chasing, stalking, pouncing, etc.) emerge separately during the juvenile period and are first displayed in a highly discontiguous fashion. In his words:

> [These] individual elements may sometimes already combine in their later form, e.g. lying in wait with stalking, the stalking run with the pounce, and so on, but at once they separate again and appear singly or in combination with other playful movements, some of which come from other functional contexts than that of prey-catching.

It is precisely this unpredictable display of behavioral patterns, in uncoordinated sequences and separated from their normal functional context, which we naturally term 'play.' While play is in one sense an epiphenomenal consequence of the metamorphosis occurring in the behavior system during the juvenile phase, it is also a condition which, as a consequence of its variability, can act as a powerful engine for opportunistic learning. As Coppinger and Smith remark, "In displaying each motor pattern in many different sequences and social situations, a young mammal is *learning* the use and social effect of that motor pattern almost as though it were a tool being manipulated" ([23], emphasis in original). The juvenile survives its metamorphosis to adulthood by being a facile learner and employing play as its primary learning tool.

2.7.2 computational implications: the importance of dynamic restructuring

Thus, it turns out that the matter of play has rather profound computational implications. Specifically, we have seen that even systems that are very well adapted to their environment (e.g. mammals) can benefit greatly from opportunistic learning. Moreover, the gradual rearrangement of behavior modules that occurs during the metamorphosis from neonate to adult can provide exactly the kind of variability and experimental fluidity that is necessary to drive this sort of learning. Opportunistic learning, in other words, is greatly facilitated by the dynamic restructuring of the behavior system and the opportunities for 'play', or the experimental deployment of behavior in novel contexts, that it provides. In the parlance of machine learning, play is an excellent means of balancing the competing demands of *exploration* and *exploitation* [54]. All of this makes a strong suggestion to the artificial intelligence community that we should not shy away from the messiness incurred in the on-line reorganization of behavior systems. Enduring the seeming disorder of a metamorphosing behavior system may allow us to design artificial creatures that adapt to the precise, unpredictable nature of their environments with much greater agility than can be achieved using more static approaches.

Let us note the connection between the point being made here and the critical assertion of the preceding corollary, that learning, even of behaviors that can be largely pre-specified, has significant advantages over building-in. Learning, like the more general processes of developmental reorganization, implies a dynamic rearrangement of the elements of the behavior system. The process of learning one skill, therefore, introduces variability into the behavior system that can potentially help to uncover opportunistic contexts for learning another skill. By designing systems that productively engage with processes of behavioral reorganization rather than attempting to minimize them, we design systems that, much like children, are capable of learning unexpected things in unexpected ways.

2.8 lesson two: learn with the right tools

There is a general bias in the engineering of complex systems towards the minimization of mechanism. Domain-general solutions, having a wider range of applicability to multiple problems, are invariably seen as preferable to more restricted domain-specific solutions. While this desire for parsimony is perfectly rational in theory, in this section I will argue that such a design philosophy may not always be optimal in practice. In particular, I will consider the problem of learning in autonomous artificial creatures, and argue that it is better approached from the standpoint of specialized rather than general mechanisms.

Even learning can be innately guided, so that a creature "knows" in advance how to recognize when it should learn something, what object or individuals it needs to focus on, which cues should be remembered, how the information ought to be stored, and when the data should be retrieved in the future.
(J.L. Gould, quoted in Hauser, 1996)

The central contention of this section is that if we want a system to learn within multiple content domains then endowing it with distinct specialized tools for each of those domains is likely to be a more powerful approach than attempting to create a single general-purpose solution. The idea is similar to that advanced by Minsky in his influential *Society of Mind* text, wherein he argues that "The power of intelligence stems from our vast diversity, not from any single, perfect principle" [52]. Of course, skeptics may argue that such an approach is highly unaesthetic, sacrificing the goal of a single elegant solution in favor of an undignified assortment of narrow 'hacks.' My counterclaim is simply that a well-designed set of special purpose tools can have an elegance of its own. If the reader will indulge a slight digression, let us consider the Swiss army knife as an example. A Swiss army knife may be a collection of distinct, domain-specific tools, but it is certainly not inelegant. Rather, it is a well-integrated solution to a variety of problems, one that respects the fact that when you have a can of soup to open the tool that you want is a can-opener, not an awkward hybrid between a knife, a can-opener, and a corkscrew. Returning to the domain of cognition, my claim is that the greater the amount of domain-specific prior knowledge that can be deployed to help constrain and guide any given learning task, the easier that task will be. Specialized learning mechanisms help to make this kind of useful structuring of prior knowledge possible.

From a developmental perspective, there is strong evidence that much of the learning that maturing creatures do is driven by highly specialized domain-specific mechanisms. Indeed, the use of specialized learning machinery during development seems to be the natural counterpart of the previously discussed tendency to learn in specialized contexts; the two key lessons of development, **learn in the right context** and **learn with the right tools,** are actually very tightly coupled. In the following I will consider some specific examples of this specialized learning phenomenon drawn from the wealth of research on birdsong learning. I will then go on to consider what these examples imply about the necessary features of a computational approach to learning specialization, and the role that such specialization might play in a developmental AI architecture.

2.8.1 biological inspiration: birdsong learning

From the human perspective, birdsong may be among the most commonly encountered and aesthetically pleasing manifestations of animal behavior. Perhaps partially because of this native appeal, birdsong has been one of the most vigorously studied sub-fields of ethology, and has given rise to a startlingly complex research picture.

[Birdsongs] are among the most highly patterned sequences of actions that animals perform naturally; only the songs of whales come close in the degree of complexity that they display.
(Marler, 1999)

The phenomenon of birdsong is also of significant special interest to developmentalists, due to the complexity of the factors that appear to control the occurrence and refinement of song during ontogeny. Of particular interest for our purposes, current research into the developmental trajectory of species-typical song repertoires has uncovered some remarkably unexpected findings, and even begun to challenge the entrenched 'learned versus innate' dichotomy. As we shall see, the development of species-typical song seems to be a process of learning in the context of "innate foreknowledge that is remarkably comprehensive" [47]; it is learning, in other words, carried out using highly specialized, domain-specific tools.

Let us begin by considering the traditional model for birdsong learning, known as the sensorimotor model [46]. In outline,[4] the model proposes that the newly hatched chick first acquires an auditory template for its species' song by listening to the vocalizations of conspecifics in its environment. Later in life (often several months later), when the young bird begins to vocalize on its own, it uses this stored template as a guide for refining its own song. While conceptually simple, this model accounts for a large body of experimental findings. Its hypothesis of auditory templating based on social experience explains the well-substantiated finding that chicks deprived of exposure to their conspecifics' vocalizations will fail to

4. In order to avoid digressing too far from the matter at hand, I here present only the briefest summary of the sensorimotor model. For further detail, the reader is referred to Marler's writings on the subject. A recommended starting point is [46].

manifest species-typical song later in life (see, for example, [49]). Similarly, the importance of the learned template for guiding later efforts at vocalization is supported by the fact that birds deafened after template formation (but before experience of their own vocalization) also produce highly abnormal song [39].

Though the weight of the experimental evidence argues that the sensorimotor model is likely to be correct in many respects, more recent thought has cast doubt on whether it represents the full story. The key insight here comes from Marler, who has argued that the kind of open learning posited by the sensorimotor model is not enough to account for the fidelity with which species-typical song is transmitted between generations [46]:

> Given the uncertainties of copy error, the likelihood of drift as song patterns are passed from generation to generation, and perturbations of the transmission process by intergenerational changes in the acoustic environment in which songs are learned, it is unlikely that any aspects of song structure would remain uniform over time throughout an entire species range if song learning were a fully open process.

In other words, the song-learning task confronted by developing birds must be being constrained in some fashion, or it could not possibly be accomplished in as successful and robust a fashion as it is observed to be. How, for example, do birds in natural habitats know which of the birdsongs in their environment they are supposed to learn? Why do chicks reliably learn their own song rather than, say, the song of a different species in a neighboring tree?

Marler and Sherman provided some insight into these questions via their comparative study of song development in swamp sparrows and song sparrows [48]. Under natural conditions, the songs of these two species are easily distinguishable, with the call of the song sparrow being longer and more complex than that of its less melodious counterpart. Note that the prediction of the basic sensorimotor model, however, is that these cross-species differences should largely disappear when chicks from the two species are raised under identical conditions. After all, if species differences are grounded entirely in differential experience with the song of conspecifics, then it should be fairly easy to undermine these differences by manipulating the environment of the developing chick. Marler and Smith tested this prediction by raising swamp sparrow and song sparrow chicks under identical conditions of social isolation. Not surprisingly, the chicks raised in these circumstances failed to develop normal songs. What was surprising, however, was the fact that the resulting songs were in fact differentiable using the same criteria used to differentiate the two species' songs in the wild. The isolated song sparrows' songs, for example, were longer, contained fewer trilled syllables and more note complexes than those of the isolated swamp sparrows, recapitulating key differences that are observed in normally raised birds from these species.

To explain these and other similar results, Marler proposed that birds must have some form of preactive template, a low-level auditory template that exists innately (e.g. irrespective of actual experience with conspecifics' song) and that specifies some of the basic attributes of the species-typical song [46]. Preactive templates are hypothesized to become active at a particular point in ontogeny, helping to focus the bird on the right portion of its 'song space' in order to make learning and refinement easier. The implication here is that birds are not approaching the song-learning task from a cognitive tabula rasa, but rather with some significant degree of prior knowledge. It is as though the bird is endowed with some general notion of what it is that needs to be learned before the learning actually occurs.

An even more striking manifestation of the role that foreknowledge may play in the song learning process regards the timing of critical periods for exposure to conspecifics' song. In this context, critical period is the term applied to the window of time in which a chick can use experience with the song of conspecifics to construct the song template hypothesized by the sensorimotor model.[5] Though findings here are not yet conclusive, work by Whaling et al. [77] suggests that chicks may actually be able to extend the critical period if they fail to receive appropriate tuition during the usual time-window. It is as though the chick is able to determine that

5. As distinct from the preactive templates which Marler proposed in his refinements of the basic model.

it has not yet encountered the appropriate learning context, and retain its receptivity to learning in consequence.[6]

Experimental findings such as these, as well as the preceding logical considerations regarding the stability of species-typical song, all militate strongly against the kind of completely open learning proposed by the basic sensorimotor model. The contrary suggestion is that "the native songbird brain has extensive foreknowledge about the natural song patterns of its species" [46]. The songbird is indeed learning about its song on the basis of auditory experience with conspecifics, but this learning appears to be taking place under the direction of a highly specialized mechanism capable of using innate prior knowledge to simplify the learning task. With these points of biological inspiration in mind, let us now turn to an examination of how this kind of domain-specific learning machinery might be adapted to a computational setting.

2.8.2 computational implications: specialized learning tools

As the birdsong examples illustrate, there is good reason to believe that much of the learning that animals do during development is greatly simplified by the deployment of specialized domain-specific mechanisms. How might we go about designing computational analogues of such

6. Of course, it is not necessary for this determination to be conscious. In the sparrows studied by Whaling et al., for example, the extension of the critical period appears to be mediated by a suppression of testosterone levels that occurs under conditions of social isolation.

mechanisms to serve a similar simplifying role in a developmental AI architecture? In this section I will articulate the conceptual specification from which the implementation of such computational analogues, which I term **specialized learning tools** (SLTs), will proceed in **chapter 4**. This articulation will take the form of a series of design requisites representing the essential features of SLTs as distilled from our consideration of biological development.

requisite 1: contextual triggers

Specialized learning tools must contain mechanisms for identifying the context or contexts in which they can operate most effectively; they must be capable, in other words, of initiating (or providing the option to initiate) the learning process that they embody when conditions are favorable for its success. Note that this requisite makes explicit the important connection between **lesson one** and **lesson two**. In **lesson one** we discussed the general features that characterize advantageous contexts for learning; the onus for translating these general considerations into specific policies for the activation of learning mechanisms is carried by the SLTs.

As was implied in **lesson one**, a given SLT may specify multiple contexts in which it could profitably be engaged. It seems likely in fact that most SLTs, particularly those corresponding to the learning of 'instinctive' or otherwise critical behavioral elements, would want to specify a diverse collection of possible learning contexts. These contexts could be assigned some prioritization that would interact with the perceived urgency of the learning objective. For example, an SLT might specify an ideal learning context, one that might be highly specific but worth waiting for so long as the content of the learning in question is not yet critical. This ideal trigger could then be supplemented with a trigger for a less optimal but more common context, one that could perhaps be utilized if the creature were running out of time in which to learn the desired content.

requisite 2: representation of task importance

In the context of a complete creature, learning is just one of many cognitive tasks that must be carried out at any given time in order to survive; learning may not always be the creature's most important priority. In order to optimally integrate learning into the creature's ongoing behavior, it is therefore important that each specialized learning tool maintain an explicit representation of the current importance of its designated learning task. This importance rating will almost certainly be dynamic, reflecting the fact that certain learning tasks will increase or recede in importance over the course of ontogeny. Through the careful specification of possible learning contexts and management of task importance ratings, it should be possible to define an intelligent balance between learning things that are opportunistically convenient, and learning things that are necessary.

requisite 3: structured prior knowledge

The most important design requisite for any specialized learning tool is the embodiment of domain-specific prior knowledge relevant to its learning task. Such prior information is the analogue of the "extensive foreknowledge" which was discovered to be present in the context of birdsong learning; it is the innate understanding of the problem in question that helps to both simplify and minimize the learning that must take place.

While the relevant prior knowledge for any given learning situation is likely to be quite problem specific, it is possible to make a few meaningful generalizations. First, it is important that an SLT embody some prior knowledge regarding the particular environmental stimuli or internal states that are likely to be most salient for its learning task. The SLT should provide some heuristics, in other words, to help identify the relevant portions of the creature's state and action spaces. At a less abstract level, we can think of this variety of prior knowledge as specifying a particular attention management strategy; it dictates how the creature should direct its perceptual resources in order to maximize the likelihood of learning.

Second, the ethological literature strongly suggests that SLTs should include a 'best guess' or default model for the content that is to be learned, something very much like the preactive template proposed by Marler in his work on birdsong learning. While one might be suspicious of detailed prior models of this sort on the grounds that they might make learning unduly rigid, the biological evidence does not support this position. On the contrary, so long as the models include a suitable provision for being overridden on the basis of specific contrary experience, it appears that they may actually make learning a good deal more robust. Additionally, detailed prior models allow the creature to actively monitor the progress of its own learning, an idea that we will take up again shortly.

requisite 4: representation of the reward signal

This design requisite gets at one of the most subtle but pervasive lessons of development: the fact that developing creatures seem to be sensitive to implicit reward signals that are far less obvious than the kinds of unconditioned stimuli we are used to considering. What, for example, is the relevant reward signal that drives the refinement phase of birdsong learning, in which the bird converges on the appropriate form for its species-typical song? All of the available evidence suggests that the 'reward' in this case is simply the goodness-of-fit between the bird's vocalization and its stored auditory template [46]. Meaningless in any other context, the bird somehow comes to construe this information as a reward signal that guides learning in the specific circumstance of song refinement.

The implication here is that SLTs should maintain a representation of the reward signal that is meaningful for their particular learning task. If such a reward signal can be specified, then many apparently unsupervised learning problems can be converted into a simpler supervised form, one in which the learning tool itself is providing the feedback. Note that this point again recapitulates the central importance of prior knowledge for learning in development. If the SLT has some approximate representation for what is to be learned (as was discussed in the last design requisite), then it can greatly simplify the learning task by providing feedback to help channel it in the right direction.

requisite 5: ability to detect sufficient learning

One highly desirable consequence of the detailed prior knowledge that has been attributed in the design of SLTs is the fact that this knowledge makes it possible to detect when sufficient learning has occurred. That is, in addition to being able to detect when conditions are appropriate to begin learning, it is important for SLTs to be able to detect when conditions support the cessation of learning.

Aside from the obvious benefits that this design requisite confers in terms of efficiency, it also makes it possible for the creature to detect circumstances in which it is failing to learn in the desired fashion. This knowledge could in turn be employed to modify other parameters of the SLT such as the target learning context or the rated importance of the learning task. If the system is failing to learn in its current state, in other words, it can take the proactive measures of attempting to utilize a different learning context, or dedicating more cognitive effort to the learning task.

requisite 6: mechanism for integration

One of the most important challenges of learning, easy to overlook in theory but impossible to bypass in practice, is the question of how the results of learning should be integrated into the behavior system. How can a learning creature make the most effective use of what it has discovered about the world? The final design requisite for specialized learning tools is that they should provide a specific mechanism for addressing this question.

While it is difficult to make very specific claims at this conceptual level of analysis, it seems logically reasonable to argue that dividing the burden of learning amongst a diversity of specialized tools may help to make knowledge integration significantly more tractable. That is, much of the difficulty of integration in the abstract comes from the diversity of the forms in which meaningful knowledge can accumulate during learning. Defining a completely generic scheme for knowledge integration is therefore unlikely to be a successful enterprise. The SLT approach to learning, however, avoids this pitfall entirely. Because each SLT is responsible for learning within just one well-defined domain, each SLT need only specify an integration policy for one kind of knowledge; this is a much more constrained problem that is likely to have a feasible solution in the majority of cases.

the big picture: specialized learning tools as behaviors

In order to conclude the discussion on specialized learning tools, let us consider the topic from the perspective of the behavior systems in which they will be contained. The important point to be made here is that SLTs can be thought of in much the same terms as any other computational behavioral primitive. In the AI architecture of Blumberg et al., for example, the behavior system is comprised of behavioral primitives known as action tuples [8]. Each tuple specifies a trigger context in which it can profitably be engaged, a particular action or motor pattern that it initiates, and a 'do-until' context that specifies when it should disengage. Note that SLTs have a high-level organization that mimics that of the action tuple very closely; SLTs have contextual triggers to signal when they can be usefully activated, a representation for the pattern of action that they embody (e.g. a particular learning strategy), and an ability to detect when they should cease to be active. Thus, in many respects, SLTs can be thought of in much the some terms as any other form of behavior; they are just behaviors that happen to have the beneficial side-effect of introducing new knowledge into the creature.

2.9 summing up: the key lessons of development

This chapter has covered quite a lot of ground with respect to the computational significance of development. Before progressing on to a review of related prior work in the AI domain, let's take a moment to briefly sum up the key computational lessons of development.

The first key lesson of development is that the 'when' of learning is often just as important as the 'how'. In particular, it has been shown that the best context for learning a given behavior will often be very different from the context in which that behavior will ultimately be expressed. Development makes this point very strongly, because the ability to identify the right contexts for learning often makes the difference between a young creature's survival and its premature demise. As corollaries to this central point, it was noted that the process of development also relies on the clever use of multiple and opportunistic contexts for learning. Important behaviors can often be reliably learned rather than simply hard-wired in advance by providing multiple, independent experiential pathways for their development. This process of learning, like the more general ontogenic processes of which it is a part, implies a structural reorganization of the creature's behavior system, which in turn supports a high degree of exploratory, opportunistic learning.

The second key lesson of development is that learning should be undertaken using the right tools. This lesson issued a direct challenge to the conventional computational wisdom that less, from a mechanistic perspective, is almost always more. Specifically, it was shown that the diverse forms of learning that occur during development are not supported by an elaborate domain-general mechanism, but rather by a plurality of highly specialized domain-specific tools. The critical feature of these specialized learning tools is that they include a significant, often surprising, degree of prior knowledge about the problem that they are intended to solve. By embodying a specific notion of what it is that they are intended to learn SLTs help make learning much more tractable, and allow it to proceed in advantageous contexts that do not provide obvious reward signals to guide the learning process.

These lessons serve as the conceptual foundation for the developmental AI architecture that is described in this thesis. Before we begin considering the specific implementational details of that architecture, however, it is important to position this thesis in relation to prior work in the computational domain. How, in other words, does a developmental approach to AI fit into the larger landscape of machine learning and artificial intelligence research? The next chapter will consider this important question.

relationship to prior computational work 3

Organizing a review of prior computational work relating to development turns out to be a surprisingly challenging enterprise. On the one hand, it is true that the idea of an explicitly developmental behavior architecture has only a handful of precedents in the literature. On the other hand, it is also true that researchers as early and eminent as Minsky [52] have long argued that the phenomenon of cognitive development represents both an important challenge and profound opportunity for AI. While such a multifaceted intellectual history is obviously amenable to multiple perspectives, I will now present my own interpretation of how this thesis' research fits into the web of prior computational work.

3.1 organization of this review

My review will begin with a brief consideration of machine learning, the field of related work that is (seemingly) most distant from the developmental themes being elaborated in this thesis. I will argue that, despite their surface differences, the concerns of machine learning and developmental AI are actually complementary in important ways. Contrastingly, I will then discuss artificial life and genetic algorithms, approaches to computational intelligence that appear on the surface to be more 'biological.' Here I will assert that the evolutionary bent of such approaches is actually much less helpful for the developmental research program than the more formal tools of machine learning. The complementarity between machine learning and developmental AI will then be more explicitly highlighted by shifting the discussion to a consideration of behavior-based AI and synthetic characters. Recent work in this domain, particularly that of Blumberg and collaborators ([6], [7], [8], [5]) provides an excellent demonstration of the convergence between the tools of traditional machine learning and more biologically-motivated approaches to AI architecture.

Having set the broader context for this thesis, I will then move on to a discussion of recent work that is more explicitly developmental in character. I will begin with the field of sociable robotics and its treatment of a number of developmental themes such as the infant-caregiver interaction and the emergence of theory of mind. This will lead me to the few extant examples of developmental behavior architectures, and a rather enlightening comparison between insightful and down-right ugly approaches to the problem. Finally, I will conclude the chapter with a brief consideration of alternative computational conceptions of development drawn from the psychological community.

3.2 a developmental take on machine learning

The field of machine learning is an impressively expansive one ([54] and [27] are recommended as good introductions) and our consideration of it here will be mainly philosophical. To wit, the key feature of machine learning from the perspective of this thesis is its algorithm-centric view of learning. Machine learning, in other words, is much less concerned with the architectural organization of learning in the context of a complete system than it is with the (often abstract) consideration of specific learning algorithms. This is of course not a bad perspective to take on the problems of computational learning, simply one that is quite different from the developmental perspective being developed here. As we will see, however, these two perspectives are actually complimentary in quite important ways.

Consider for example the subfield of reinforcement learning (detailed in [38] and [67]), a common technique for allowing computational systems to learn from experience. In brief, reinforcement learning considers an agent, capable of executing a discrete set of actions, that exists in a world whose possible configurations can be described by a set of discrete states. The goal is for the agent to learn which actions to take in which world states in order to achieve some desired goal or reward state [7]. While we will return to reinforcement learning in more detail later in this chapter, the salient point for now is simply that there are many aspects of cognitive development that can be formally reduced to a problem of this form. Berlin, for example, has described the development of predatory behavior in felids as being formally equivalent to a reinforcement learning problem [5]. In other words, reinforcement learning is a potentially powerful *tool* that can be deployed in the context of a developmental *architecture* in order to solve a specific problem.

The point I am leading up to here is simply that the concept of the specialized learning tool maps beautifully onto the methodological framework of machine learning. We have described SLTs as domain-specific computational mechanisms capable of solving particular problems. SLTs can thus be conceptualized as simply architectural 'wrappers' for particular machine learning algorithms. A developmental creature might have a reinforcement learning-based SLT for guiding the development of stereotyped behavioral sequences, a neural net-based SLT for learning to recognize conspecifics, and so forth. The algorithmic concerns of machine learning are thus highly complementary to the architectural concerns of developmental AI.

3.3 artificial life and genetic algorithms

Moving in the direction of more biologically inspired prior work, let us now consider the fields of artificial life (ALife) and genetic algorithms (GAs).[1] Investigators in these fields have studied the construction of systems that develop in an *evolutionary* (as distinct from *ontogenetic*) sense. Ray, for example, investigated the evolution of small computer programs that competed for the limited resource of CPU time, and was able to demonstrate the emergence of adaptive strategies such as parasitism [58]. Hammond and collaborators created a simulated ecosystem in which simple agents mated, fought, and ultimately evolved within a resource-limited environment [36]. Sims showed how genetic algorithms could be used to evolve functional morphologies for locomotor tasks such as walking and swimming [64].

While on the surface this work may appear to be somewhat more relevant to developmental AI than machine learning, I argue that this is in fact not the case. The reasons for this relate to the timescale and complexity of the learning pursued in ALife and GA systems. Note that work in these

1. Genetic algorithms are of course themselves an example of machine learning. For the purposes of this discussion, however, I wish to draw a distinction between formal and stochastic machine learning approaches. Genetic algorithms being of the latter variety, I will be content to group them with artificial life without further comment.

fields is generally concerned with how a *population* of agents adapts over time; how, in other words, does natural selection take place within a virtual world in order to favor the survival and transmission of particular behavioral traits? Because of this emphasis on populations rather than individuals, ALife and GA approaches typically utilize simple agents and stochastic learning mechanisms. The complexity, in other words, comes not from the behavior of any particular agent, but rather from the aggregate behavior of the population as a whole over time. By contrast, the emphasis of the developmental AI agenda is on driving sophisticated learning processes within the context of a single creature's behavior system. The goal is not to create a horde of mindless agents that randomly converge on an adaptive behavioral strategy across hundreds of generations, but rather to create a single 'smart' creature that can learn what it needs to learn in real-time without making any catastrophic errors. While ALife and GA research has generated interesting and useful insights, I argue that the algorithmic rigor of machine learning combined with the kind of architectural scaffolding being developed in this thesis is a more productive platform for pursuing the goals of developmental AI.

3.4 behavior-based ai and synthetic characters

Though it has not in general had an explicitly developmental emphasis, behavior-based AI is the subfield of existing computational research to which this thesis has the strongest intellectual connections. I will accordingly treat this topic in more depth, beginning with a brief historical sketch of its development before moving on to a consideration of its current state-of-the-art.

3.4.1 a historical interlude

The historical trajectory of behavior-based AI is arguably best traced to Brooks' work on autonomous robotics. Drawing inspiration from creatures such as insects, Brooks articulated the notion of a *subsumption architecture* in which a hierarchy of simple computational modules (each decidedly non-intelligent on its own) gives rise to coordinated, intelligent top-level behavior [17]. Brooks' approach was revolutionary in that it produced robots capable of navigating robustly in real-time, in stark contrast to the torturously slow pace of prior autonomous robots [18]. Aside from the specifics of the subsumption architecture (which would go on to influence researchers from Minsky [52] to Blumberg [6]), the key idea in Brooks' work was that simple behavioral rules, akin to those that animals seem to employ to guide their actions, could often outperform more formal mathematical approaches to AI.

This idea was appropriated into the realm of computer graphics by Reynolds in his much-cited paper on the animation of avian flocking behavior [59]. Reynolds used agents called 'boids', governed by a simple perceptual mechanism and behavioral ruleset (e.g. avoid collisions, match velocity and remain near to other boids), to animate flocking without painstakingly specifying the motion of each individual. Though their capabilities were modest, the architecture of Reynold's boids was an important watershed: they were individually self-contained entities, situated in a virtual world and making decisions about their behavior in real-time on the basis of their perceptions. They were, in other words, simple but complete synthetic characters rather than abstract disembodied agents.

Reynolds work was followed and significantly expanded by Tu and Terzopoulos, who created intricate underwater animations using autonomous virtual fish [72]. Tu and Terzopoulos provided a good explicit statement of behavior-based AI's defining characteristic, remarking that their approach to simulating natural complexity was "to model each animal *holistically* as an autonomous agent *situated in its physical world*" (emphasis mine). Tu and Terzopoulos' fishes had an intriguing physics-based locomotor model, but more important from our current perspective was their behavior system. Much elaborated relative to Reynold's set of simple behavioral rules, Tu and Terzopoulos' behavior system included perceptual input from multiple sensors, representations for "habits" (e.g. individual behavioral preferences) and drives such as fear, hunger, and libido, and a loosely hierarchical organization of behavioral routines. Though the initial emphasis of Tu and Terzopoulos, like Reynolds, appears to have been the animation properties of their creatures, it is clear from their enthusiastic paper that they become increasingly intrigued by the behavioral sophistication of their creations. Situating complete synthetic characters in meaningful virtual environments, in other words, was proving to be a promising approach to AI as well as computer graphics.

Following Tu and Terzopoulos's fish was Blumberg's (rather more charismatic) dog Silas ([9], [6]). Blumberg's work made the fundamental contribution of taking the emerging behavior-based AI paradigm and putting it on a solid biological footing by considering the computational insights to be gained from classical ethology. While references to biological ground-truth had by now begun to appear in the computer animation literature ([29], [50]), they were largely absent from work such as that of Tu and Terzopoulos.[2] By contrast, all aspects of Blumberg's system, from the hierarchical organization of the behavior system to the perceptual model and action selection strategy, were modeled after the ethological theories of investigators such as Baerends, Dawkins, Lorenz, and Tinbergen [6]. Blumberg's work on Silas thus marks the origin of the computational ethological perspective that undergirds this thesis.

2. Tu and Terzopoulos do make passing reference to the animal behavior literature, but the details of their behavior system owe more to common sense intuitions about 'what fish seem like they ought to do' than to ethology.

3.4.2 current work: the synthetic characters group

With this historical context in view, let us now consider some of the more recent work in behavior-based AI, and its relationship to the developmental research program of this thesis. While behavior-based AI has now percolated into domains as diverse as interactive cinema [55] and consumer electronics [1], I wish to focus on how the ethological perspective that Blumberg pioneered with Silas has continued to be refined in the recent work of the Synthetic Characters Group at the MIT Media Lab. This work provides an excellent example of how the formal tools of machine learning can be combined with insights from ethology to yield powerful results, and thus serves as an important model for this thesis.

a computational model of clicker training

One of the most intriguing combinations of ethologically-inspired AI and machine learning to come out of the Synthetic Characters Group is a clicker-trainable virtual dog named Dobie [8].[3] Note that the problem faced by Dobie, that of learning 'when to do what' in order to maximize the reward received from a human trainer, is fundamentally a problem of reinforcement learning. Formally, Dobie must discover the state-action pairs that lead to the most reliable pattern of reward in his environment.

On the surface it might seem that this problem could be solved through the straight-forward application of an 'off-the-shelf' reinforce-

3. See Pryor [57] for a detailed discussion of clicker training.

ment learning technique such as Q-Learning ([74], reviewed in [67] and [5]). In summary form, Q-Learning represents the state-action space as a grid, with rows corresponding to perceptual states and columns to possible actions. The Q-Learning algorithm is essentially an incremental, exhaustive search of this state-action space in which the learning agent gradually converges on the appropriate value for each state-action pair given the pattern of reward it experiences. While the algorithm is theoretically sound in that it is guaranteed to converge [67], it is simultaneously nearly useless for a 'real-world' situation such as that faced by Dobie.

Note that Dobie's potential state and action spaces are both continuous and extraordinarily large. Even assuming that one could impose some approximation of discretization on these spaces, their sheer size would make Q-Learning intractable. What is needed is a more clever variant of Q-Learning, and that's where ethology comes in. Real dogs of course solve the learning problem that Dobie faces, and do so with lemon-sized brains that have considerably less computational horsepower than the computers on which Dobie is implemented. As Blumberg and collaborators have argued, the secret to this feat is that real dogs know how to take advantage of reliable heuristics and trainer feedback. For example, dogs (as well as other animals) are biased to learn proximate causality: they constrain their search for the cause of a reward to a narrow temporal window around the occurrence of the reward ("I got a cookie because I sat immediately after my owner said 'sit', not because I was scratching my ears five minutes ago"). Similarly, dogs make excellent use of the supervisory signals provided by the trainer to guide their exploration of their state, action, and state-action spaces. While we will not digress into the full details of the implementation here (the interested reader is referred to [8]), the upshot is that by building these sorts of ethologically motivated heuristics and biases into Dobie, Blumberg *et al.* were able to create a trainable computational system with many of the same learning capabilities as a real dog.

a computational model of predatory sequence development

Another line of research from the Synthetic Characters Group that relates to this thesis is Berlin's recent investigation of the problem of sequence learning. Again combining the methodological perspectives of machine learning and ethology, Berlin utilized models of the development of predatory sequences in felids (see, for example, [43]) to implement a novel variant on Q-Learning termed *timed hierarchical Q-learning* (THQL). THQL was shown to be extremely effective at learning to sequence behaviors in order to achieve a distal reward. Interestingly, the fundamental insight of the algorithm was essentially **the first key lesson of development**, that the 'when' of learning is as important as the 'how'. By staging the onset of different behavioral capabilities in a staggered fashion and motivating his creature to experiment with new skills as they were acquired, Berlin was able to improve significantly on the performance of the standard Q-learning algorithm. Berlin's work is thus a very good example of how ethological insight can not only complement but actually improve traditional machine learning techniques.

to sum up

The critical point that I wish to drive home here is fundamentally the same as the conclusion of **section 3.2: a developmental take on machine learning**: by combining the formal tools of machine learning with architectural insights provided by ethology, extremely powerful results can be obtained. The work in this thesis is simply an extension of this principle into the domain of development.

3.5 sociable robotics

Thus far we've examined how this thesis relates to the broader context of non-developmental prior work, from traditional machine learning to behavior-based AI and synthetic characters. With this background in place we can now turn our attention to work with a more explicitly developmental flavor. To begin, I'd like to consider the recent burgeoning of research in the field of sociable robotics.

Defined in large part by the work of Breazeal, the goal of the sociable robotics research program is to develop robots that "model people in social and cognitive terms in order to interact with them" [13]. The ease of interaction that sociable robots strive for is beneficial not only from the perspective of interface design, but also with respect to the robot's own learning. That is, by interacting with people in an intuitive fashion that is evocative of a child-caregiver or student-teacher interaction, such robots can take advantage of our natural capacity for teaching. Note that this idea of using innate *social scaffolding* to drive the learning process ([11], [14], [15]) is an extremely close parallel to the prior chapter's notion of embedding structured prior knowledge in domain-specific learning tools; both lines of theory emphasize the importance of innate structure in helping to make learning in complex environments tractable. Not surprisingly then, ideas from cognitive development have been central to the sociable robotics research program from its inception [19], and indeed have recently inspired the establishment of a new subfield termed 'epigenetic robotics' [56]. In this section I will confine my attention to the main-line of sociable robotics research as represented by Breazeal and Scassellatti's work, examining how these researchers have made excellent use of insights drawn from specific phenomena in the broad sweep of cognitive development. The rather more questionable results of the epigenetic robotics research program will be considered briefly in the following section.

Undoubtedly the best known sociable robot is Breazeal's Kismet [11]. With its expressive face and blue eyes, Kismet is designed to interact with humans in a fashion that explicitly mimics the dynamics of the infant-caregiver interaction [16]. For example, much as infants help to regulate the duration and intensity of their social interactions by providing easily discernible affective feedback to their caregivers (e.g. fussing or looking away), so does Kismet provide similarly intuitive feedback to help the user interact at a distance that is optimal for its vision system. Kismet can also use a combination of affective feedback and proto-speech to entrain users in a turn-taking 'dialogue' analogous to that which caregivers engage in with their pre-linguistic infants [12]. Scassellatti has undertaken related investigations focusing on slightly different dimensions of the infant-caregiver interaction. For example, Scassellatti's work has addressed how psychological theories of joint attention and theory of mind in infancy might be applied in order to give robots these critical social abilities ([60], [61]).

This type of sociable robotics research has been exceptionally fruitful. For the purposes of this thesis, however, we should note that its approach to development differs from the more architectural perspective that I have begun to develop in this thesis. Whereas work in sociable robotics has focused on implementing specific models of individual human social competencies and their development, it has not yet addressed the broader question of how insights from development might be used to structure the organization and learning abilities of a general-purpose behavior system.

3.6 developmental behavior architectures

Let's now turn to an examination of the prior work that is closest to the content of this thesis: the construction of developmental behavior architectures. As has been said, there has actually been only a modest quantity of prior research directed at this specific object. We'll examine two approaches: the Autonomous Mental Development research program of Weng *et al.*, and Drescher's schema-based architecture.

Hailing from the aforementioned epigenetic robotics camp, Weng and his collaborators have purported to be interested in the broader architectural perspective on development that concerns this thesis. Pursuing a research program that they term Autonomous Mental Development (AMD), Weng *et al.* have called for the creation of robotic systems whose behavior is determined by "developmental programs" that will allow them to be raised to be competent at tasks that have not been exhaustively prespecified [75]. Unfortunately, this work seems to be based almost entirely on the flawed intuitive formulation of development as a process of linearly increasing complexity that was discussed in the previous chapter. Weng *et al.* make virtually no reference to the actual biological phenomenon of development, and consequently have no concrete proposal for how the developmental programs they describe might be implemented. While the most recent results of the AMD program are interesting from the standpoint of neural networks ([37], [76]), they make no headway on the larger issue of designing developmental behavior architecures.

A much more credible foray into developmental architectures was provided by Drescher's investigation of a computational learning system inspired by the theories of the famed psychologist Jean Piaget [26]. Drescher's system is based on a core representation which he terms a *schema*. At its simplest level, a schema can be thought of as a specification of a pre-condition, a post-condition, and an action that connects the two with a specified degree of certainty. For example, a schema might represent the notion that *if* the sidewalk is icy *and* I run to catch the bus *then* I am likely to end up with a new bruise. Of course, the schemas that Drescher is concerned with are much less complex than this, pertaining to the basic sensorimotor learning of a simple virtual creature in a highly constrained 'gridworld.' What is remarkable about Drescher's architecture is the fact that within this simple environment, his creature's learning was shown to recapitulate some of the early milestones in Piagetian cognitive development. Beginning with a handful of extremely primitive percepts (e.g. the ability to detect an object in a given region of the field of view, the ability to sense contact on a given portion of the body), Drescher's creature was able to bootstrap its way to the formation of much more elaborate concepts such as object permanence. This bootstrapping was achieved using a clever mechanism for generating radically new concept representations using what Drescher refers to as *manifestations*, or cumulative perceptual states.

Drescher's work is quite elegant but not without its flaws. In particular, as Blumberg has remarked, the fact that Drescher's creature had "no focus of attention, either spatially or temporally" [6] meant that its learning was constrained to occur through a process of "exhaustive cross-correlation" [26]. That is, in order to determine which actions might be relevant to which consequences (without the benefit of any *a priori* constraints) Drescher's system was continually forced to compare *all* of the creature's actions with *all* of the perceived changes in the environment across the *entire history* of the simulation. Though Drescher argued that this computational burden was acceptable, it proved to be enough to rapidly bring down a Connection Machine modeling a *7 x 7* gridworld [6]. Scalability, obviously, was not a strength of Drescher's architecture.

Foner and Maes [28] attempted to remediate this weakness using an approach quite similar to the temporal windowing mechanism previously described for Blumberg's Dobie [8]. By giving a Drescher-esque developmental creature a focus of spatial and temporal attention and the ability to use that focus to restrict the search for meaningful action-consequence correlations, the efficiency of the learning was improved by as much as 30 times [6]. Again the application of ethological principles, in this case the heuristic of proximate causality, has proved a fine complement to a more traditional machine learning/AI approach.

Before leaving this section, it should be noted that Drescher's work differs from this thesis both in its targeted scale of application and its central methodological commitments. In terms of scale, Drescher is concerned with understanding and simulating only a very narrow slice of ontogenetic time – that of early infancy. Also, as we have seen, scalability to complex environments is a significant (though not necessarily insurmountable) challenge for Drescher's system. Methodologically, it is worth noting that Drescher is firmly on the side of domain-general learning architectures, and that he largely rejects the importance of innate structure that has been argued for in the preceding chapter. Thus, while Drescher's work represents an excellent counterpoint to this thesis, it has not covered the same intellectual ground that I intend to tread.

3.7 alternative computational conceptions of development

Finally, as a conclusion to this chapter, it is interesting to consider a radically different computational conception of development drawn from the psychological community: the dynamical systems approach of Thelen and Smith ([68], reviewed in [42]). Thelen and Smith oppose what they see as the overly zealous application of nativism in the mainstream of developmental psychology and have thus attempted to articulate an altogether different view of developmental organization. What makes this work worthy of mention in a computational literature review is the mathematical and computational rigor of their alternative conception. Collaborating with physicists and mathematicians, Thelen and Smith have attempted to describe a broad range of developmental phenomenon, from motor development to aspects of higher-level cognitive processes, as arising from self-organizing complex dynamical systems. In common with connectionists, Thelen and Smith subscribe to a view of developmental organization in which the distinction between representation and algorithm (or "structure" and "process" as they put it) is collapsed. Thelen and Smith view cognition not as the result of explicit computation over a set structured representations, but rather as a consequence of the emergent and distributed interaction of simple units that have no independent representational significance. While Thelen and Smith's ideas are very much outside of the mainstream in developmental psychology (and, for that matter, rather far afield from the central conceptual commitments outlined in this chapter) they are interesting to consider as an alternative means of conceptualizing the computational aspects of development.

Having thoroughly surveyed both the biological ground-truth of development and the computational work that it has inspired, we are now ready to turn our attention to the specific implementational details of this thesis. In the next chapter we will embark on a discussion of the developmental behavior architecture that represents this thesis' core computational contribution.

a developmental behavior architecture 4

In the previous chapter we examined the biological literature in order to get beyond the intuitive myth of development. Rather than a simple linear progression in complexity, we found development to be a process through which creatures *reconfigure* their complexity in order to adapt to their changing ontogenetic environment. The critical connection between development and learning was examined in detail, and the key lessons of **learning in the right contexts** and **learning with the right tools** were emphasized. In this chapter we will deploy these insights in order to design a developmental behavior architecture.

4.1 design preliminaries

Let's begin by considering how the key computational lessons of development can be translated into specific design guidelines for the architecture.

In particular, I'll now show how the conclusions of the preceding chapter can be mapped onto the design principles of hierarchy, modularity, and distributed learning.

4.1.1 hierarchy

Our intuitive conception of behavioral organization naturally lends itself to hierarchy. That is, we tend to think of our lower-level behaviors (for example, slicing bread) as being coordinated by superordinate levels of behavioral structure (I sliced bread *because* I wanted a sandwich *because* I was hungry). Ethological theorists too tend to ascribe hierarchical organization to behavior. The previously cited behavior systems proposals of Baerends, Blumberg, Dawkins, Hogan, Timberlake, and Tinbergen ([1], [6], [24], [25], [70], [34], [69], [35], [71]), for example, all rest on the foundation of hierarchical structure.

In addition to its intuitive tractability, hierarchical behavior organization has several important computational advantages. First, it is conceptually simpler to define behavior selection strategies when the behavior space is divided according to its granularity; behavior selection in a hierarchical system can proceed incrementally from the general to the specific rather than having to arbitrate across an expansive flat space. Second, a hierarchical behavior system allows learning to occur at multiple levels. We might have motor learning occurring at the lowest level of the behavior system, that of the primitive motor action, while the sequence learning necessary to coordinate multiple motor actions occurs at a higher level of the hierarchy. For all of these reasons, the developmental behavior architecture developed in this thesis will be organized in a hierarchical fashion.

4.1.2 modularity

The subject of modularity arose previously in **section 2.3**, where we discussed its relationship to the kinds of discontinuous behavioral changes that are observed during development. Recall that modularity is envisioned as providing a conceptual mechanism for large-scale developmental reorganization via the 're-wiring' of connections between behavioral modules. As this idea has proven an appealing and powerful theoretical construct, we will seek to embed the flexibility of modular construction in the design of the behavior system.

To be slightly more specific, I will take the modularity design principle as dictating that each element of the behavioral hierarchy should be a complete and independent unit of functionality capable of generating a specified behavioral consequence. Moreover, these elements should be as uniform in their external interfaces as possible such that they can dynamically reconnected and reconfigured. This kind of flexibility will be quite important for implementing the **first key lesson of development**, that of giving our creatures the ability to deploy skills in different contexts from those in which they were originally learned.

4.1.3 distributed specialized learning

This design guideline is a direct echo of the **second key lesson of development**: we wish to bypass a monolithic general-purpose learning mechanism in favor of a multiplicity of domain-specific **specialized learning tools** (SLTs). While I will not here rehearse the benefits of this learning architecture which were extensively discussed in the preceding chapter, it is worth pointing out that this organization has some important engineering benefits. Specifically, by dividing a creature's learning requirements into individual specialized mechanisms, it becomes easier to integrate those mechanisms with the relevant nodes in the behavior system. Distributing learning, in other words, simplifies the process of unifying the learning process with the creature's behavior. The modular nature of SLTs also allows vastly different learning algorithms to operate simultaneously in the context of a single behavior system. We will return to these matters in greater detail later in this chapter, when we begin discussing specific implementational details for SLTs.

4.1.4 summing up the preliminary design

Our preliminary design discussion has resolved that the behavior system will be comprised of a hierarchically organized set of behavioral modules and associated specialized learning tools. In the next two sections of this chapter, I will discuss the design of these components in detail.

4.2 behavioral elements

The modules of the behavior system, its constitutive 'atoms', are **behavioral elements.** In this section I will examine the common core of engineering that unifies all behavioral elements and provides for their reconfigurable modularity, as well as the specializations found in the various behavioral element subtypes.

4.2.1 the basics

organization

The behavior system consists of a set of behavioral elements arranged in a hierarchical lattice. Each behavioral element is allowed to have multiple parents as well as multiple children. A critical feature from the standpoint of modularity is the fact that none of the behavioral elements has any specific representation of its parents and children; the current view of all the system's parent-child relationships is instead centrally maintained by the behavior system. Hence, each behavioral element can operate as a reasonably independent entity; the element adheres to a communication model to exchange information with whatever parents and children it may have, but it does not represent specific information about the state or identity of these neighboring elements. From the perspective of the individual behavioral elements then, a change in the hierarchy's overall structure would be entirely transparent.

communication

Behavioral elements communicate with one another through a message passing protocol. Messages called **behavioral parameters** can be exchanged between parents and children on every tick of the simulation. While the contents of behavioral parameters can vary widely, the following are the most commonly exchanged data:

- **modifiers and specifiers** for the current behavior. For example, a higher element in the behavior system could potentially pass an 'adverb' to its child in order to modify the style of the final motor output (e.g. "whatever you do, you should do it happily").
- **propagated value**, which can be used for altering the course of action selection (**section 4.2.3**).
- **feedback from SLTs**, which allows learning to influence the expression of behavior (**section 4.3.2**).

The role that these kinds of data play in the behavior system's functioning is elaborated in later sections of this chapter.

4.2.2 subtypes

The complete behavior system is composed of behavioral elements of several different subtypes inspired by the hierarchical levels recognized in Timberlake's theory of behavioral organization [69]. As has been said, these subtypes all share a considerable amount of common engineering; however, they also have some important differences that allow them to serve distinct roles in the behavioral hierarchy. I will now briefly review the three major subtypes of behavioral element.

motivational subsystems

At the highest (e.g. most abstracted) level of the behavior system we have **motivational subsystems**, which map directly onto Timberlake's notion of *subsystems* [69]. As the name implies motivational subsystems can be thought of as the entry points for distinct subhierarchies of the overall behavior system, each corresponding to a particular motivational state. Because they are the highest level of the behavior system, motivational subsystems play a special role in behavior selection which I will discuss in **section 4.2.3**.

behavioral modes

The children of motivational subsystems are generally **behavioral modes,** which correspond to Timberlake's *mode* constructs [69].[1] Behavioral modes can be loosely thought of as 'strategies'; they correspond to patterned sequences of goal-directed actions. For example, a creature with a *hunger* motivational subsystem might have *hunt* and *scavenge* as two associated behavioral modes, each represents a particular sequence of motor behaviors useful for achieving the common motivational goal.

1. The behavior architecture makes very few restrictions as to which subtypes of behavioral elements are allowed to be the parents/children of other elements. The only definitive rule is that the 'roots' of the behavior system must be motivational subsystems, while the leaves must be motor primitives.

motor primitives

Behavioral modes generally have **motor primitives** as children, corresponding to Timberlake's *actions* [69]. Motor primitives are where the proverbial rubber meets the road in the behavior system; they correspond to individual atoms of motor activity, and provide an interface point for the creature's motor system. Motor primitives are also the leaves of the behavior hierarchy and do not themselves have children.

4.2.3 behavior selection

The process by which behaviors are selected for expression is one of the most important aspects of any computational behavior architecture. In this thesis' behavior system, behavior selection proceeds from roots (motivational subsystems) to leaves (motor primitives) in a fashion that maximizes behavioral coherence while preserving the creature's ability to react rapidly to unexpected changes in the environment.

value and activity states

Before commencing with a specific discussion of the behavior selection process, we first need a bit more information about the representation of behavioral elements.

Behavior selection is value-based, with higher-valued behavioral elements winning expression over those which are less valued. Hence, each behavioral element continuously updates an explicit representation of its **value**, roughly its expected utility given the current state of the world.

Value is computed from both intrinsic and extrinsic factors. Intrinsically, each behavioral element has a fixed base value. Extrinsically, behavioral elements may receive **propagated value** from children, parents, or associated SLTs. As we will see in section **section 4.3.2**, this value propagation is central to the mechanism by which SLTs influence behavior selection.

As behavior selection proceeds, behavioral elements are assigned different activity states. At any given moment, behavioral elements may be either **passive**, **active**, or **focal**. A behavioral element is **focal** if it is the current locus of behavior selection; note that only one element of the behavior system may be focal at a time. Those behavioral elements which are not focal but are on the path of current behavior selection (e.g. were previously focal but then passed focus on to one of their children) are said to be **active**. All other behavioral elements are **passive**. Note that this definition of activity states means that the focal element is always connected to an active motivational subsystem through a sequence of active intermediate elements, an idea that is illustrated in **figure 4.1**.

moving forward: motivation to motor action

Now that we've seen the representational machinery that supports behavior selection, let's move on to consider the *forward* portion of the behavior selection process. In forward selection, the creature refines a motivation to an appropriate motor action. To accomplish this, on each step of forward selection the focal element arbitrates amongst its children, selecting the

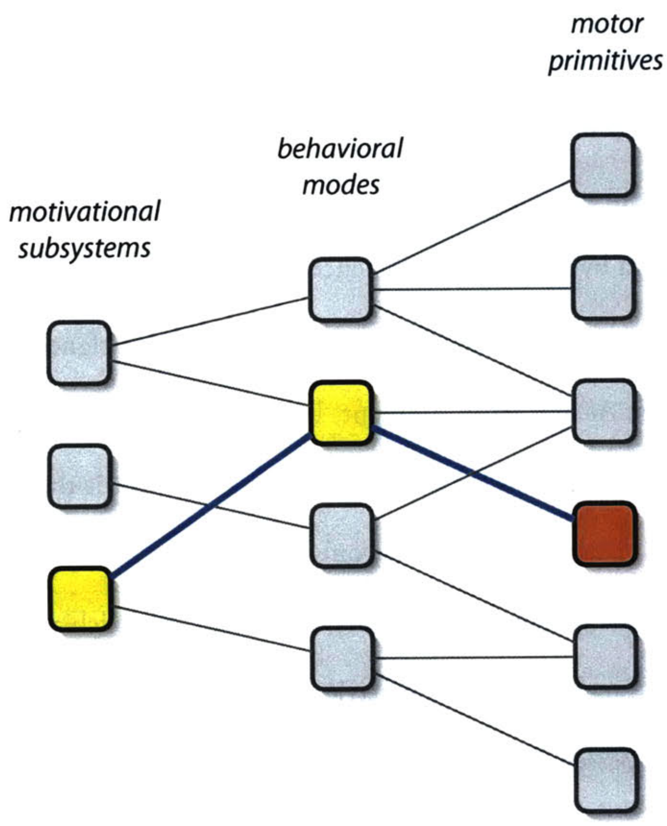


figure 4.1 The focal element (shown in orange) is always connected to the active motivational subsystem through a string of active behavioral elements (shown in yellow). The route traced between the two is referred to as the **path of behavior selection** (shown in blue).

child with the highest value. This child then becomes focal, and the formerly focal parent reverts changes to the active state. This process recurs until a motor primitive becomes focal, an event that corresponds to the expression of the corresponding action. A schematic illustration of this process is provided in **figure 4.2**, which depicts forward selection from a motivational subsystem to a motor primitive.

moving backward: maintaining coherence

Once behavior selection has reached a motor primitive, the behavior system must step *backwards* in some fashion in order to select a new behavior.

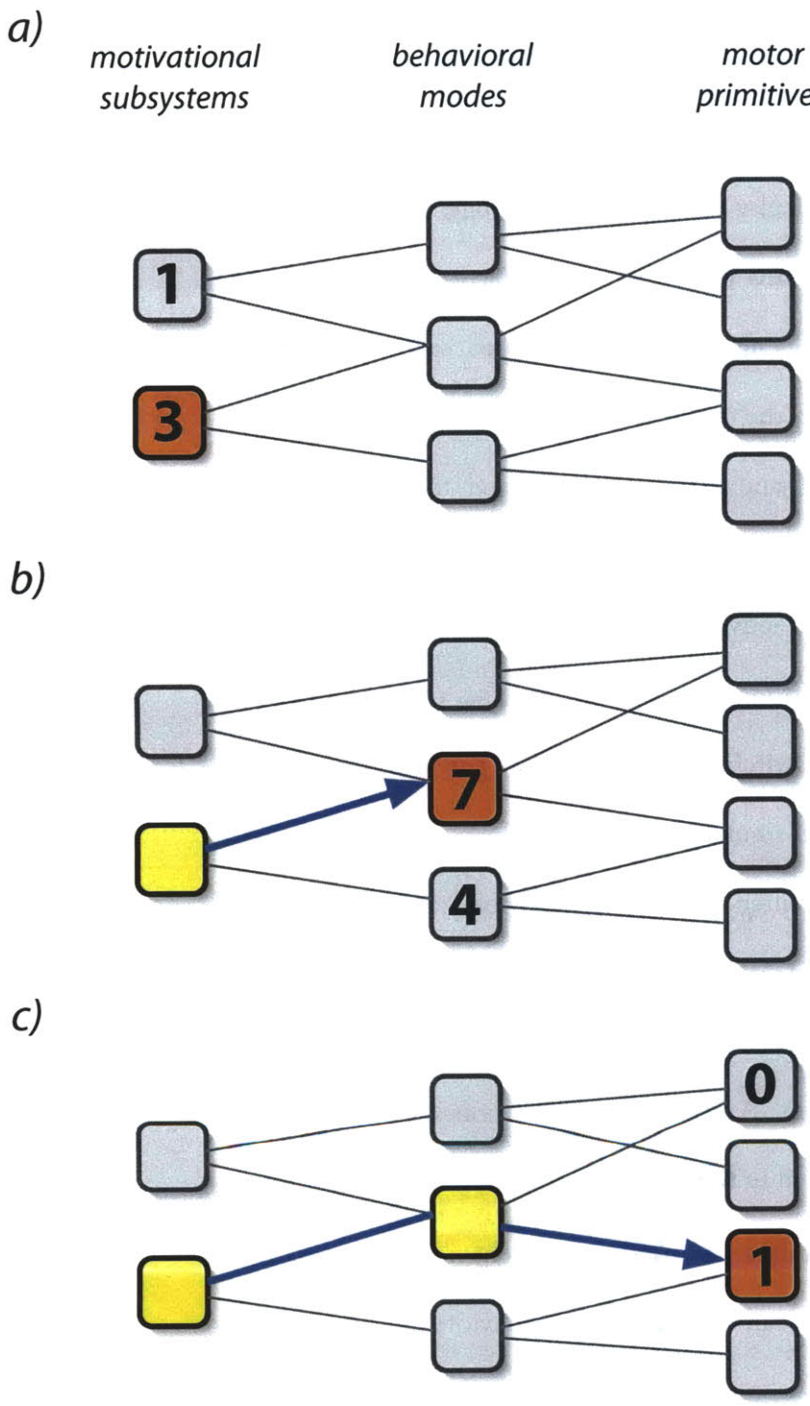


figure 4.2 An illustration of forward behavior selection, in which focus shifts from a motivational subsystem to a motor primitive. In (a), arbitration amongst the motivational subsystems has identified the highest valued and made it focal. In (b), this focal subsystem proceeds to arbitrate its children and make the highest valued amongst them focal in turn. This process iterates until a motor primitive is reached, at which point the creature expresses the corresponding action. Note that the binary values present during motor primitive arbitration illustrate the effect of a sequencing constraint in the parent behavioral mode.

Note that there are several possible approaches to making this backwards step. For example, we might consider making the currently active motivational subsystem focal and effectively restarting behavior selection from that point. The difficulty with this methodology, however, is that it presents a challenge in terms of maintaining behavioral coherence. If a creature is attempting to engage in a coordinated sequence of motor actions, then the greater the number of behavior selection steps that intrude between *action A* and *action B*, the greater the likelihood of a pathological departure from the desired sequence. Note that moving backwards all the way to the active motivational subsystem maximizes this number of behavior selection steps. By contrast, this thesis' approach to backwards selection is to switch focus to the *active parent* of the currently focal element. The critical result of this methodology is that once a focal motor primitive has executed, its active behavioral mode parent will become focal.[2] This behavioral mode, in turn, will shift behavior selection back into the forward direction, making the next motor primitive in its represented sequence focal. A schematic illustration of this backwards stepping process is provided in **figure 4.3**.

Of course, motor primitives aren't the only behavioral elements from which backwards stepping can be initiated. A behavioral mode, for exam-

2. Of course, as has been mentioned before, a motor primitive need not *necessarily* have a behavioral mode as its parent. This is simply the most common case, and hence the most useful for explanatory purposes.

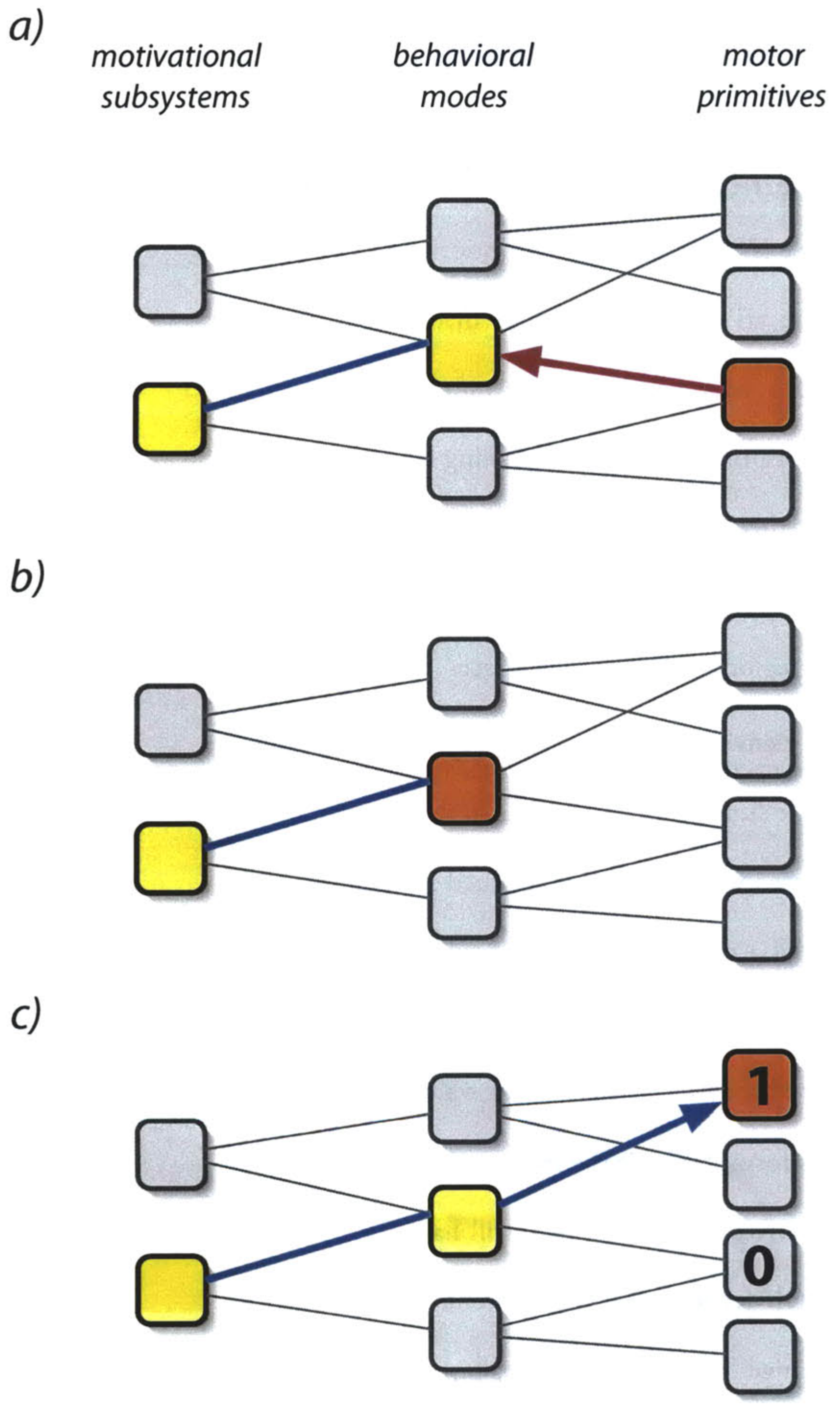


figure 4.3 An illustration of the backwards portion of behavior selection. In **(a)** the focal motor primitive has fired, and is returning focus to its active parent. Note in **(b)** that because this backwards stepping shifts focus up just one level in the hierarchy, the coherence of sequences represented by the focal behavioral mode may be preserved maintained. Thus, in **(c)**, the second motor primitive in the sequence represented by the active behavioral mode becomes focal.

ple, that has reached the end of its prescribed sequence of motor primitives might cause behavior selection to step backwards to its active motivational subsystem.

startling: maintaining agility

Though behavioral coherence is often an important priority, sometimes it is more important to react to an unexpected change in the environment immediately [21]. If you put your hand on a hot stove, you want to remove it pronto rather than calmly putting the finishing touches on your grilled cheese sandwich. For this reason, motivational subsystems can initiate a **startle** response that causes the creature to rapidly transition between arbitrarily disjunct regions of its behavior space.

Unlike other behavioral elements, motivational subsystems recompute their value on every tick of the simulation. In particular, they continuously update the extrinsic propagated value which has accrued from all of their child elements. If the value of a passive motivational subsystem surpasses that of the currently active motivational subsystem, then a startle occurs and control transitions immediately to the newly high-valued subsystem.

A critical element in this formulation is the fact that even when behavioral modes and motor primitives are passive, they still have the opportunity to continuously dispatch feedback to their parents. Thus, when a passive element detects a change in the world that makes it suddenly more valuable, it can dispatch the appropriate propagated value signal upward through the hierarchy to its parent motivational subsystem(s). Motivational subsystems, then, act as clearinghouses of sorts for behavioral recommendations from all of their children. When the combined value of these recommendations suggests that the creature should reevaluate its current behavioral priorities, the startle mechanism insures that exactly that occurs.

Now that we've had the opportunity to become thoroughly acquainted with the design of behavioral elements, let's move on to the matter of learning. In the next section, I'll explain how specialized learning tools are integrated into the behavior system architecture that has been described thus far.

4.3 specialized learning tools

Inspired by the discussion of domain-specific learning mechanisms in **chapter 2**, learning in the behavior system is accomplished using a distributed collection of **specialized learning tools**. As we will see in this section, many of the design principles for SLTs intentionally mimic those employed in behavioral elements, thereby allowing for a seamless integration between acting in the world and learning from those actions.

4.3.1 the basics

Let's begin by considering some of the foundational aspects of SLT design.

organization

Like behavioral elements, SLTs are designed to be arranged in hierarchical structures; an SLT hierarchy is termed a **toolbox.** Toolboxes are intended to take advantage of the naturally hierarchical nature of many learning tasks. While a toolbox's component SLTs are generally each capable of operating independently, their integration into the toolbox allows them to *share and integrate* the results of their learning in a way that helps to focus and simplify learning at higher and lower levels.

As a conceptual example, consider a character that is learning to play checkers. At the simplest level the character must first learn the basic mechanics of the game: the ways in which pieces are allowed to move and how to capture the opposing player's pieces. Building on these basics, the character must then begin learning about the best way to coordinate effective sequences of moves. At a higher level still, the character also needs to begin to learn about his opponent's strategy and the best means of countering it. A **specialized learning toolbox** for such a character could incorporate individual SLTs designed to address each level of this learning hierarchy. The learning output from lower-level SLTs in the toolbox would help to constrain and refine learning at higher levels of the hierarchy. We will return to this notion in more detail in **chapter 5**, where we will see how a toolbox hierarchy helps to make a multi-faceted learning objective more tractable.

communication

In order for SLTs to cooperate they must first communicate. This is accomplished using the same message passing protocol employed in behavioral elements. SLTs can exchange **learning parameters** with their parents and children that may contain data such as:

- **activation and deactivation signals** used to control SLT activity.
- **perceptual data** that has been filtered or refined using the results of an SLTs learning.
- **results** from hypothesis testing used to refine predictive models.

Again, we will see more specific examples of this communication process in **chapter 5**.

In addition to communicating with one another, SLTs also need to be able to communicate with behavioral elements. This is accomplished by dispatching behavioral parameters to the communication channels of target elements, a subject that we will now explore as part of the SLT theory of operations.

activity states

Like behavioral elements, specialized learning tools can be either active or passive at any given time. These activity states are designed to help conserve computation, as only active SLTs participate in the potentially expensive process of forming and testing hypotheses described in the next section. The intuition is that if the learning context for which an SLT was designed is not present, it should wait in a passive state that requires very little computational effort. Indeed, the only thing that passive SLTs do is to execute a simple update routine on each tick of the simulation designed to determine whether they should become active. For the remainder of this chapter we'll confine our consideration to the operation of active SLTs.

4.3.2 theory of operations: hypothesize-act-update

With the basic aspects of SLT design now in place, we may now turn our attention to a description of their pattern of operation. SLT operation is partitioned into **learning epochs**, or iterations of a hypothesize-act-update cycle.[3] During each epoch, *i)* an SLT forms a hypothesis as to how the state of the world is likely to evolve (generally conditional on some suggested action), *ii)* the creature optionally performs the suggested action, and *iii)* the SLT attempts to update its predictive model with the observed consequences. Using this readily generalizable framework, SLTs are capable of addressing a wide variety of specific learning challenges. Let's consider each portion of a learning epoch in detail.

hypothesis formation

At the outset of each learning epoch the SLT must form a hypothesis as to how it thinks the state of the world will evolve over the next (arbitrary) period of time. This hypothesis formation is generally accomplished by

3. It should be noted that my hypothesize-act-update cycle is conceptually quite similar to the expectation generation mechanism used by Burke to learn apparent temporal causality relationships [20].

combining data from *i)* the currently perceived state of the world, *ii)* previously perceived state stored in the belief system, and *iii)* the results of prior hypothesizing stored in the SLT.

Hypothesis formation is almost always accompanied by the dispatching of an associated behavioral parameter to the SLT's target behavioral element(s). That is, if an SLT hypothesizes that the creature will find a cookie in the cupboard, then an accompanying behavioral parameter must be formed to suggest the 'look in cupboard' action to the behavior system. This kind of behavioral suggestion can take several forms. In some instances, a behavioral parameter will literally contain an identifier for a motor primitive that the SLT is requesting be engaged. In other cases, the parameter will contain propagated value intended to bias behavior selection in a particular direction. The specific form of the suggestion generally depends on the subtype of behavioral element that the SLT is targeting with its feedback; suggestions made at higher levels of the behavioral hierarchy have more latitude for generality than those made at lower levels.

taking action

Once an SLT has formed its hypothesis it is up to the behavior system to take the action appropriate for verifying it. Note that this verification is ***not*** mandatory; learning is just one of the things that a creature will be attempting to do in the world at any given time, so behavioral suggestions from SLTs must be constantly balanced against other behavioral priorities.

In general, the more urgent it is that the hypothesis be verified, the more propagated value will be associated with the corresponding behavioral parameter.

After the SLT has formed its hypothesis and dispatched the appropriate parameters to the behavior system, it commences to monitoring the creature's current perceptual state for an **epoch completion event**. An epoch completion event indicates that the creature has taken whatever action is necessary to verify or disprove the current hypothesis and that new data is available for updating the SLT's predictive model accordingly. Returning to the previous cookie example, the firing of the 'open cupboard' motor primitive might represent an epoch completion event for the SLT in question. Note that since the behavior system is never obligated to act on the suggestions made by SLTs, many SLTs specify a maximum period of time during which they will await an epoch completion event; if epoch completion is not observed during this time period, the SLT will simply abort the current epoch and return to hypothesis formation.

updating the model

When an epoch completion event is observed, the SLT can update its predictive model using the observed results in the world. Note that this is the point in the learning epoch at which learning actually occurs. That is, it is at this point that the SLT adjusts its expectations (and hence future hypotheses) in accord with the observed consequences of the creature's actions.

4.3.3 whence specialization?

The description of SLTs in this section has focused on their core engineering – the common design traits that all SLTs share. Specialization occurs in the implementation of specific instances of SLTs, at which point the user must define the mechanisms by which hypotheses are generated, validated, and integrated into the continuing operation of the SLT. In the next chapter we will consider these matters in more detail through the example of **Hektor** the mouse, and his penchant for high-stakes card games.

hektor: a computational case study 5

It's now time to put the developmental behavior architecture described in **chapter 4** to the test. In this chapter I introduce **Hektor** the mouse, and describe how this thesis' developmental architecture will allow him to succeed in a complex competitive learning interaction with a human user.

5.1 introducing hektor

Hektor the mouse is the brother of Max (**figure 5.1**), star of a previous installation by the Synthetic Characters Group [4]. Hektor's major motivation in life, like that of his big brother, is his insatiable desire for cheese. As Hektor also has a bit of a gambling streak, he has been known to attempt to satisfy his appetite by playing a competitive card matching game called **CheeseWhiz** with a human user.

5.1.1 cheesewhiz

CheeseWhiz is a high-stakes card game in which Hektor can win cheese by wagering on the agility of his learning; a typical CheeseWhiz gaming table is depicted in **figure 5.2**.

CheeseWhiz is played with a deck of nine different cards, of which both the player (Hektor) and the House (the human user) have an unlimited supply. As shown in **figure 5.3**, each card has two features: *color* (either red, green, or blue) and *shape* (either circle, triangle, or square). At the outset of the game, the House specifies a set of matching rules by first selecting one of these two features as the **critical feature**, and then selecting a matching **response** for each possible **seed** within that feature. For example, one set of matching rules might look like the following:

Critical Feature:	color	
red	**is matched by**	blue
green	**is matched by**	red
blue	**is matched by**	green

Note that if desired, the same response can match multiple seeds.

Once the House has set the initial rules, the game is ready to begin. The player specifies a wager corresponding to the number of pieces of cheese that he would like to bet on the outcome of the upcoming hand.

figure 5.1 Hektor's brother, Max. Sharing a strong mutual resemblance to their mother, the two are said to be practically indistinguishable.

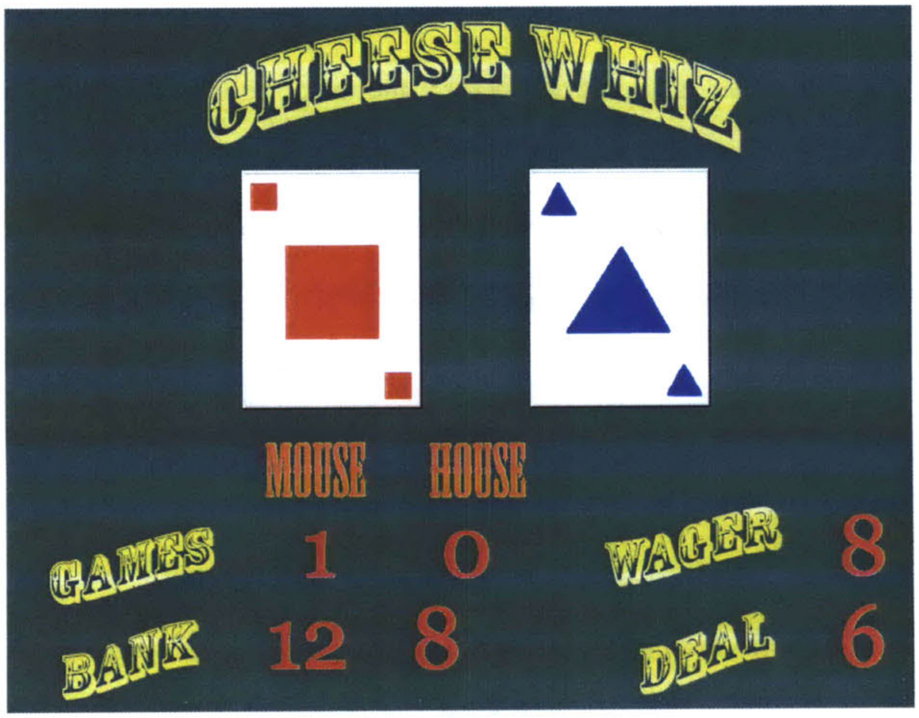


figure 5.2 A typical CheeseWhiz gaming table.

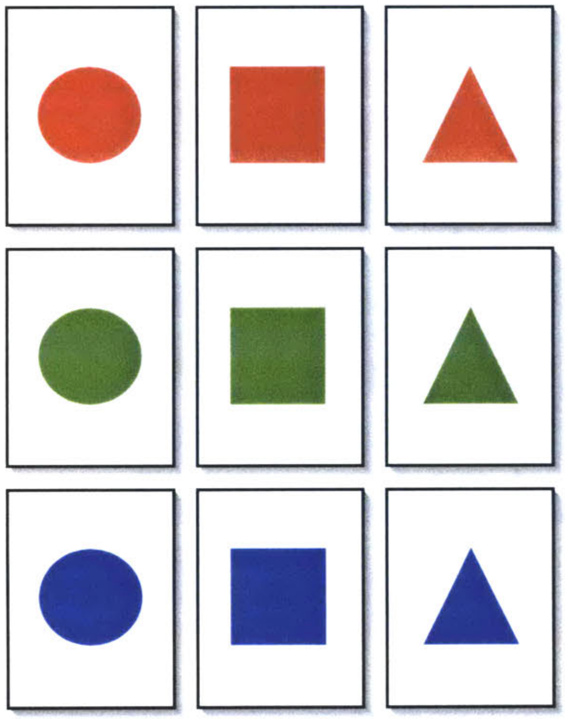


figure 5.3 The nine cards that comprise the CheeseWhiz deck.

The House then randomly deals a card, and the player must respond with a card that he thinks matches the House's play. If the player is able to match the House's card successfully he wins the wagered amount of cheese; otherwise it's time to pay up. Both the player and the House begin with ten pieces of cheese in their respective banks; a game is considered complete when one or the other party has no cheese left in the bank and 'busts'. For the purposes of this simulation we'll presume that both the player and House banks reset to ten pieces of cheese between games.

As specified thus far, CheeseWhiz is neither terribly exciting nor very challenging. A simple approach for the player would be to wager a very small amount of cheese on each hand while performing an exhaustive exploration of the possible rule space. Whatever losses might be incurred at the beginning of the game could be more than made up for once the complete set of matching rules was known. What prevents this trivial solution from being viable, and consequently makes the game interesting, is the fact that the House periodically[1] has the ability to **juggle**, or secretly modify, the matching rules. Hence the player that attempts to explore the rule space through a slow brute-force search will rapidly discover himself bereft of his cheese. The key to succeeding at CheeseWhiz is to deploy a clever learning strategy that allows the matching rules to be learned with maximum rapidity, such that they can be exploited before the House has the opportunity to change them. Conversely, when the matching rules do change, the player must be able to adjust and relearn with suitable quickness.

5.1.2 why is this a good test case?

There are several considerations that recommend CheeseWhiz as a compelling test case for this thesis' developmental architecture. First, the time constraints on learning that have been noted above mimic the fact that learning in nature must often take place very efficiently. Just as a defenseless duckling needs to learn the appropriate response to the maternal alarm

1. The minimum number of deals between juggles is variable, and can be adjusted to change the difficulty of the game. See **chapter 6** for a detailed consideration of how the interval between juggles impacts Hektor's performance.

call almost instantaneously, so does Hektor need to learn using the minimum possible number of trials. Second, as we will explore in more detail in the coming sections of this chapter, CheeseWhiz provides an excellent opportunity to mobilize the key lessons of development: there are multiple behavioral contexts between which the results of learning must be flexibly repurposed (**section 5.2.1**) as well as a multifaceted learning objective that can be profitably tackled using a set of specialized learning tools (**section 5.3**).

5.2 hektor's behavior system

In this section I will consider the organization of Hektor's behavior system. Starting with a decomposition of the behavior space into its component contexts, I will describe how Hektor's behavior is generated from the interaction of three motivational subsystems and their associated behavioral modes and motor primitives. This discussion will set the stage for the second half of this chapter, in which I will detail the specialized learning tools underlying Hektor's learning.

5.2.1 contextual breakdown of the behavior space

Beginning with the discussion of the **first key lesson of development** in **chapter 2**, this thesis has emphasized the importance of the 'when' of learning. The behavioral context in which a creature attempts to learn a given skill will often have as much to do with success or failure as the specific learning algorithm employed. Accordingly, the first important task in describing Hektor's behavior system is to divide his behavior space into its component contexts, or conceptually isolable subspaces. What, in other words, does Hektor need to be able to do exactly, and how is that set of behavioral possibilities most naturally partitioned?

One natural decomposition of Hektor's behavior space is as follows:

- **Passive Observation.** The set of behaviors involved in responding to the House's deal with a random card and observing the result. Hektor might be in this behavioral context at the very outset of the game, when he has no information with which to form useful hypotheses about the matching rules.
- **Active Exploration.** The space of behaviors needed to investigate specific hypotheses as to the game's rules. Hektor might engage in such behaviors after several rounds of passive observation when he has accumulated enough initial data to form meaningful hypotheses.
- **Knowledge Deployment.** Behaviors involved in utilizing a well-formed set of matching rule hypotheses to win (or, as we will see in the case of feigning, lose) a given hand. Hektor might engage these behaviors once he has formed a complete model of the matching rules or whenever it is important to achieve a particular outcome on a given hand.

We'll take these three subspaces, observation, exploration, and deployment, to be the set of behavioral contexts that span Hektor's possible actions.

5.2.2 behavior system organization

Now that we've identified Hektor's behavioral contexts we can progress to discussing the organization of his behavior system, depicted schematically in **figure 5.4**. In this section we'll unpack the details of this diagram.

motivational subsystems

In **section 4.2: behavioral elements** we identified motivational subsystems as the entry points for motivationally distinct subhierarchies of the overall behavior system. Having partitioned Hektor's behavior space into three distinct contexts, it is thus logically sensible to dedicate a motivational subsystem to each. Accordingly, **figure 5.4** shows observational, experimental, and deployment motivational subsystems at the top-level of Hektor's behavioral hierarchy. Consistent with the role that motivational subsystems were previously described to play in behavior selection (**section 4.2.3**), these motivational subsystems provide the overall direction as to which of Hektor's behavioral contexts is currently active.

behavioral modes

Each of Hektor's motivational subsystems has a child behavioral mode. Recall from **section 4.2.2** that behavioral modes correspond to patterned sequences of goal-directed actions. In this case the sequences of actions represented by the different behavioral modes are all identical, at least on the surface: all three modes implement a *wait-place wager-play card* sequence. The difference between the behavioral modes arises from the

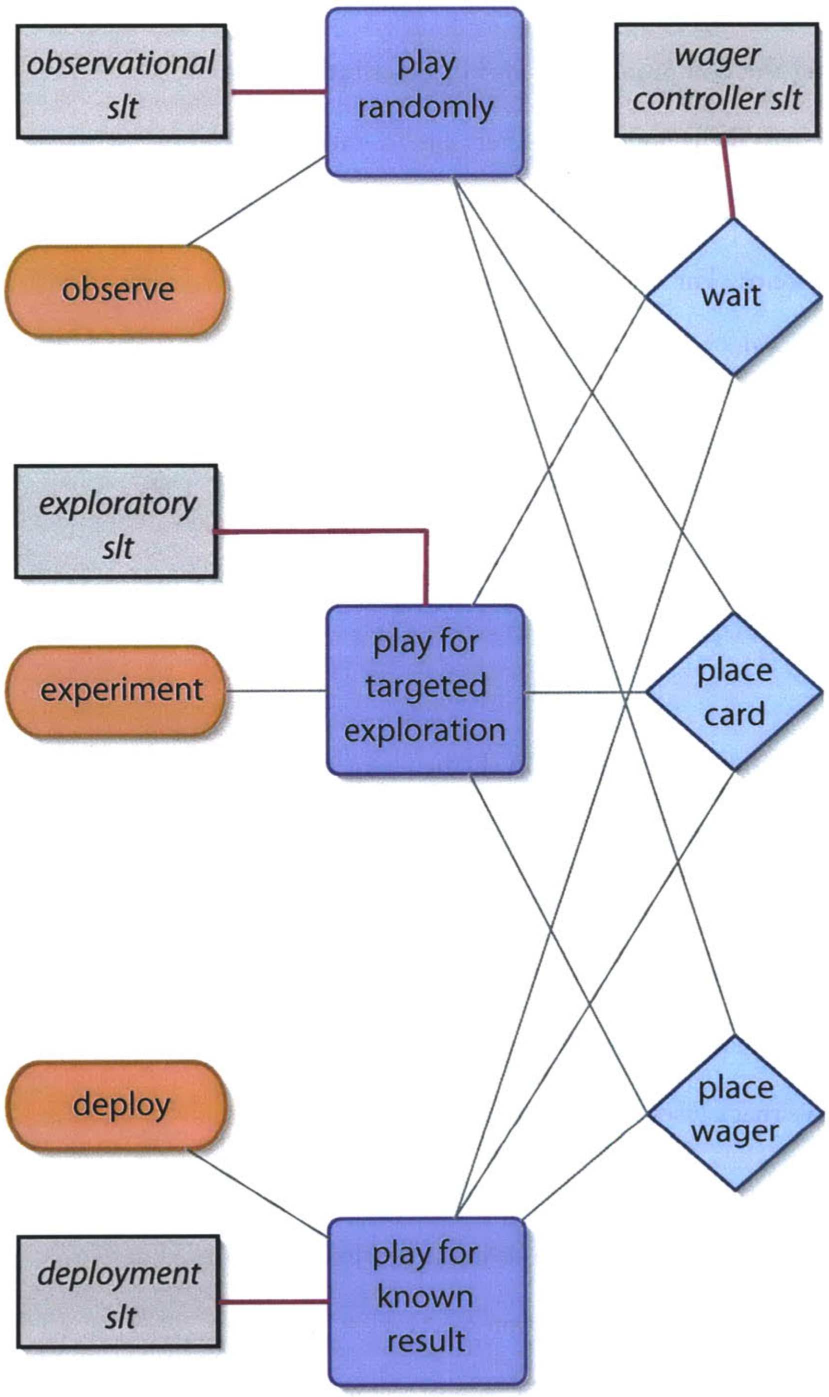


figure 5.4 A schematic illustration of Hektor's behavior system. Motivational subsystems are shown as orange ovals, behavioral modes as purple rounded rectangles, and motor primitives as blue diamonds. Specialized learning tools are depicted as gray rectangles. The gray lines connecting behavioral elements indicate parent-child relationships in the behavioral hierarchy. Red lines connecting specialized learning tools to behavioral elements indicate the points at which the SLTs communicate with the behavior system.

fact that they also serve as the interface point between Hektor's SLTs and his behavior system. Using the communication system described in **section 4.3.2**, Hektor's SLTs pass behavioral parameters on to their target behavioral modes that are intended to achieve the distinct objectives of the parent motivational subsystems. For example, the deployment SLT will pass behavioral parameters to the 'play for known result' behavioral mode indicating which card should be played in order to achieve a desired outcome on the current hand. The mechanisms by which the SLTs derive the parameters that are passed to the behavioral modes will be considered in detail in the latter half of this chapter.

motor primitives

As has been said, motor primitives are the point in the behavior system at which individual atoms of observable motor action are output by the creature (**section 4.2.2**). Hektor has a very simple set of motor primitives: an idle (default) primitive that is engaged while waiting for the House to play its card, a wagering primitive that corresponds to the act of placing a bet, and a card playing primitive that causes Hektor to put a specified card into play.

Note that this parsimony in motor primitives is made possible by two previously described engineering decisions (**section 4.2.1**). First, because behavioral elements are allowed to have multiple parents, a single set of motor primitives can subserve all three motivational subsystems. Secondly, the use of behavioral parameters allows the behavioral modes to customize the specific behavior of the wagering and card playing motor primitives; these primitives are parameterized by their active parent with the amount of the bet or the identity of the desired card respectively before firing.

With this basic view of Hektor's behavior system in place, let's now turn to the more interesting matter of how the activity of this system is orchestrated by Hektor's specialized learning tools.

5.3 hektor's specialized learning tools: an introduction

Hektor's learning is organized using four distinct specialized learning tools coordinated by a top-level toolbox; this SLT structure is illustrated schematically in **figure 5.5** and unpacking its details will occupy the remainder of this chapter. Before embarking on this discussion, however, let us first pause to make an important point with respect to the scope and limitations of Hektor's learning.

As has been emphasized, the primary concern of this thesis is *architecture*. I have set out to demonstrate how the key computational lessons of development can be harnessed to define a novel means of organizing a computational learning system. The importance of Hektor, and in particular of Hektor's learning mechanisms, is to provide a computational 'proof-of-concept' for the viability of this developmentally inspired design.

Given this (and also the inevitable time constraints of a Master's thesis), it should be acknowledged that some of the algorithmic aspects of Hektor's learning hierarchy have not been pushed to the limits of their possible refinement. In particular, as we will see in the coming sections of this chapter, both the observational and the wager controller SLTs are implemented in a simplified form, one that conforms to the architectural commitments of SLTs but does not actually learn about or adapt to the House's play patterns. While possible refinements for these SLTs will be discussed in the context of future work, the salient point at present is how *all* of Hektor's SLTs, from the simplified to the more fully-realized, together succeed in bringing the SLT architecture to life and demonstrating its potential. To put it another way, the point that I will demonstrate in the remainder of this thesis is not that Hektor's SLTs are at present a *perfect* means of addressing his learning challenges, but rather that they are an impressively good means of doing so with the *architectural potential* to become even better.

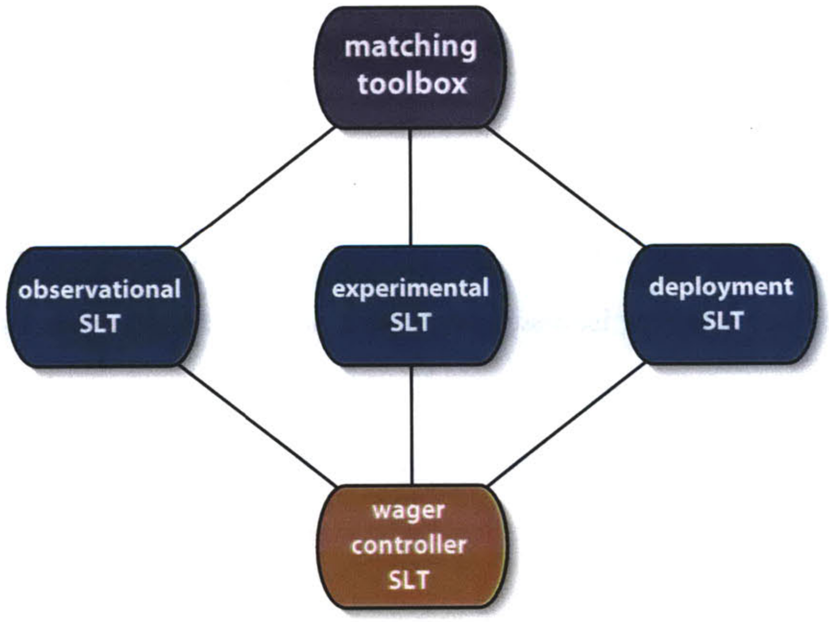


figure 5.5 Schematic diagram of Hektor's SLT hierarchy. Each of the elements of this hierarchy is described in detail in the following sections of this chapter.

With this minor caveat in place, let us now forge ahead with the detailed discussion. I will first consider the matching toolbox and its role in managing the predictive models utilized by all of Hektor's SLTs. Each of these SLTs will then be considered in turn, focusing on how their specific operations map onto the **hypothesize-act-update** cycle of the SLT learning epoch.

5.4 the matching toolbox

As discussed in **section 4.3.1**, matching toolboxes are intended to facilitate the efficient sharing and integration of data amongst a hierarchy of related SLTs. In the particular case of Hektor's matching toolbox, this facilitation takes the form of managing a set of **matching models** that are referenced and updated by all of the subsidiary SLTs. I will now briefly discuss the purpose and representational details of matching models, and then move on to the more substantive matter of how the matching toolbox goes about managing them.

5.4.1 matching models

Matching models are the basic units with which Hektor represents his hypotheses as to the game's rules. A schematic illustration of the key components of a matching model is provided in **figure 5.6**. As shown in the figure, matching models consist of:

- **A feature identifier.** Which of the two possible matching features, shape or color, is the focus of this model?

- **A representation of the putative rule set.** What are the specifics of this model's hypothesis regarding the game rules?
- **A prediction accuracy history.** Within a small temporal window, how accurate has this model been at predicting the observed course of the game?

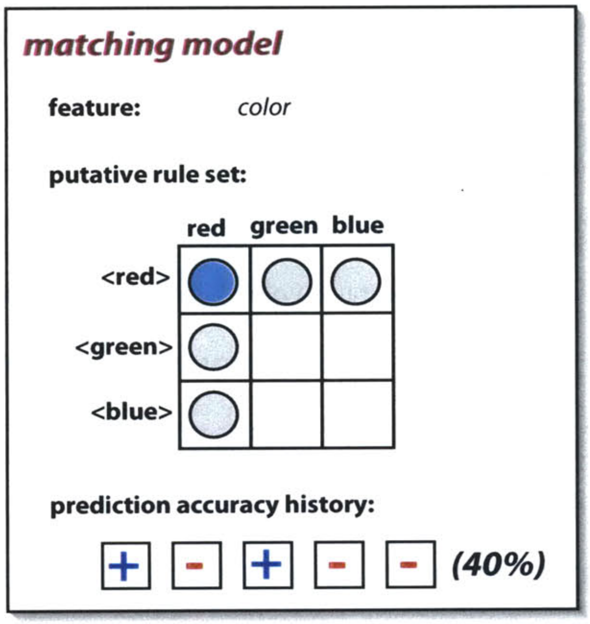


figure 5.6 Schematic illustration of a matching model. The feature specifier indicates which of the two possible matching features this model is focusing on. The putative rule set, depicted as a grid, shows the predicted results of matching seed features (whose names are listed in angle-brackets) with response features. These results are represented as either a blank grid square (no data), a gray circle (no match), or a blue circle (match). The prediction accuracy history represents this model's predictive power over a moving window of the past five deals. In the figure, deals on which the model made (or would have made) the correct prediction are indicated with a plus sign, whereas incorrect predictions or predictions that could not be made due to lack of data are indicated by a minus sign. The matching model in the figure has predicted two of the 5 past deals correctly, for a current accuracy of 40%.

5.4.2 model management

Now that we know what matching models are, let's consider how the matching toolbox goes about managing them. This management begins with the division of the models into two categories: active models and inactive models.

Active models are those that represent currently viable hypotheses as to the matching rules. At the outset of the simulation the set of active models is initialized with two empty default models, one of which presumes that shape is the critical feature and the other of which focuses on color. Subsequently, active SLTs in the **update** phase of their **hypothesize-act-update** learning epoch (see **section 4.3.2**) will signal the matching toolbox to update the active models with the outcome of each newly completed hand. If a new outcome is consistent with or enlarges upon the set of outcomes a model has previously recorded, then that model remains active. If however an observed outcome *contradicts* the contents of a model (indicating that either the model's rule hypothesis was inaccurate or that the matching rules have been juggled) then the model is deactivated and transferred to the set of inactive models.

Inactive models are models that have been found to be inconsistent with the current set of matching rules. Of course, the easiest approach to dealing with inactive models would be to simply discard them. However, as we will see, by storing inactive models Hektor gains the ability to quickly recognize when the House is reverting to an old rule set that he has previously observed. The stored set of inactive models thus helps contribute to the agility with which Hektor can respond to changing matching rules.

SLTs in the **update** phase of their learning epoch do not cause the matching toolbox to update inactive models' rule sets with the outcome of each new hand, but they *do* cause the toolbox to update inactive models' prediction accuracy histories. These updated prediction accuracy histories are used to determine when an inactive model may have returned to correspondence with the current matching rules (that is, when the House may have reverted to an old rule set represented by the inactive model) and the model should thus be reactivated. How exactly this determination occurs turns out to be a more subtle question than one might think.[2]

The most intuitive policy would be to simply reactivate an inactive model whenever its prediction accuracy history exceeded that of the most accurate active model. However, because newly initialized active models necessarily have low prediction accuracy histories until they have gone through at least some refinement, this straight-forward approach is ineffective. Reactivating inactive models strictly on the basis of a comparison between their prediction accuracy history and that of the active model(s) inevitably leads to new active models being thrown out in favor of questionable inactive models. This problem can be allayed somewhat if a pre-

2. Indeed, we will revisit this matter in much greater detail in **chapter 6**.

diction accuracy threshold, below which inactive models are ineligible for reactivation, is utilized. However, an even more principled solution is to allow reactivation to occur only when *all active models have been found to be inconsistent*. That is, the matching toolbox will revert to an inactive model only when it has conclusive evidence that the matching rules have been juggled.

For example, consider a situation in which the House opens with a **same color** rule set in which each color is matched by the same color. After a short period it switches to a different rule set, causing the **same color** matching model that had begun to represent the same color rule to be deactivated. Then, after another short period, the House reverts to the same color rule. Suppose further that the currently active model *fails to detect this change* for several hands. This could occur because, as we will see in **section 5.6: the exploratory slt**, the exploratory SLT frequently utilizes the input of inactive models when determining which card Hektor should play; the inactive **same color** model could hence by helping to steer Hektor's responding in the right direction even while the active model made incorrect predictions. As this process occurs, the matching toolbox is quietly recomputing the prediction accuracy history of the **same color** model after each hand. Since this accuracy is computed over a short temporal window, after several hands it will reach a significant value. Then, once the active model is finally proven inconsistent and deactivated, the toolbox can reactivate the existing **same color** model. Hektor can now pick up where he left off with respect to this rule set, and converge on a complete hypothesis much more quickly.

5.4.3 model generalization

The matching toolbox performs one more important function whenever an SLT causes the set of active matching models to be updated with a newly observed outcome. Specifically, the toolbox will **generalize** the active matching models whenever possible.

Generalization is based on the rules of CheeseWhiz, which state that the House must specify one matching response for each possible seed. Thus, when a matching response is found for a given seed, the toolbox can make the generalization that all other responses for that seed must *not* match. Conversely, if two non-matching responses are found for a given seed, then the only remaining response must be the match. These two forms of generalization, termed positive and negative respectively, are illustrated in **figure 5.7**.

At this point we've become familiar with the matching toolbox and its management of Hektor's matching models. Note, however, that we've been careful to point out how this management process is driven by control signals from active SLTs in the **update** phase of their learning epochs. Let's move on then to consider each of Hektor's SLTs, and the phases of their particular learning epochs, in detail. In order to map this discussion

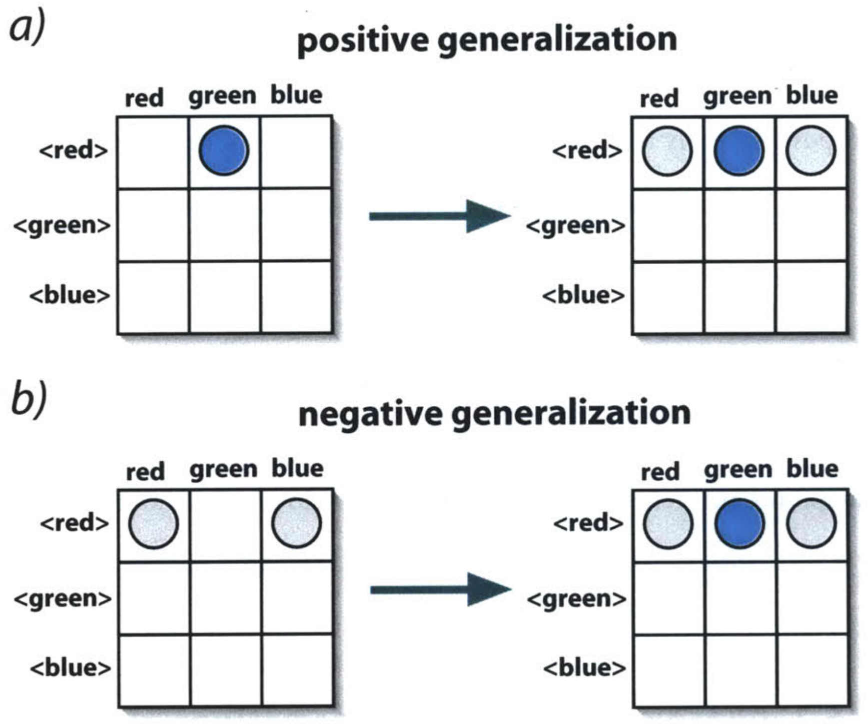


figure 5.7 Generalization of an active matching model can occur in two ways. In positive generalization **(a)**, the determination of the matching response for a given seed allows the other two possible responses to be marked as non-matches. In negative generalization **(b)**, finding two non-matching responses for a given seed identifies the third response as the match.

as explicitly as possible onto the general architectural discussion of the preceding chapter, I'll divide my commentary on each SLT into sections corresponding to *i*) its activation conditions, and *ii*) the **hypothesize-act-update** phases of its learning epochs.

5.5 the observation slt

The observation SLT is the simplest of Hektor's learning mechanisms. It's purpose is simply to 'kick-start' Hektor's matching models, generating initial datapoints that can be used to begin the more principled process of active exploration.

5.5.1 activation conditions

The observational SLT becomes active whenever the average number of datapoints per active matching model falls below a threshold value. This activation triggers the beginning of a learning epoch, the phases of which we will now consider.

5.5.2 phase one: hypothesis formation

As noted in the introduction to Hektor's SLTs (**section 5.3**), the observation SLT is one of the simplified portions of Hektor's overall learning mechanism. As such, the SLT forms no explicit hypotheses. Rather, during the hypothesis formation phase of its learning epoch it simply dispatches a behavioral parameter containing propagated value to the behavior system's observational behavioral mode.

5.5.3 phase two: taking action

Propagated value from the observation SLT causes the observational behavioral mode to become active, at which point it prompts Hektor to respond to the House's deal with a random card. The results of this random play are assimilated during the update phase.

5.5.4 phase three: updating

The observational SLT passes the consequences of playing a random card up the hierarchy to the matching toolbox, which then updates the matching models as previously described (**section 5.4.2**). Once the active models have acquired initial data in this fashion, the more interesting and proactive learning operations managed by the exploratory SLT can commence.

5.6 the exploratory slt

In contrast to the observation SLT, the exploratory SLT is the most complex and fully realized SLT in Hektor's learning arsenal. It has the responsibility of enabling Hektor to explore the game's matching rule space efficiently enough to win.

5.6.1 activation conditions

The exploratory SLT becomes active whenever the observation SLT becomes passive. In terms of the behavior system, this means that the exploratory SLT will be active whenever *either* the exploratory or deployment motivational subsystems are engaged. This, it will turn out, has important consequences for the exploratory SLT's ability to cooperate with the deployment SLT, a topic that we will consider in **section 5.7.7** and (in more detail) in **section 6.2**. Once activated, the exploratory SLT cycles through learning epochs structured as follows.

5.6.2 phase one: hypothesis formation

We can think of the exploratory SLT as forming hypotheses of the form:

> Responding to the House's deal with card ***x*** will most efficiently expand my knowledge of the matching rules.

The real meat of the computation that occurs during hypotheses formation, therefore, is the determination of this best response card ***x***.

The logic used to identify the response card can be divided into several cases depending on the number and activity state of Hektor's matching models. Each of these cases is illustrated in **figure 5.8** and discussed separately below.

multiple active models

In the event that there are multiple active models, the exploratory SLTs objective is *to attempt to differentiate between them as quickly as possible.* In other words, the SLT wants to determine which of the active models actually corresponds to the current matching rule set and deactivate the remainder. The most efficient means of accomplishing this task is by attempting to find response cards for which the active models make **conflicting predictions.** To illustrate this idea, consider **figure 5.9**.

In the figure, Hektor has thus far played three hands. On one of these hands Hektor was able to match the House's **blue square** card with a **green triangle** card. The active matching models consequently represent two pos-

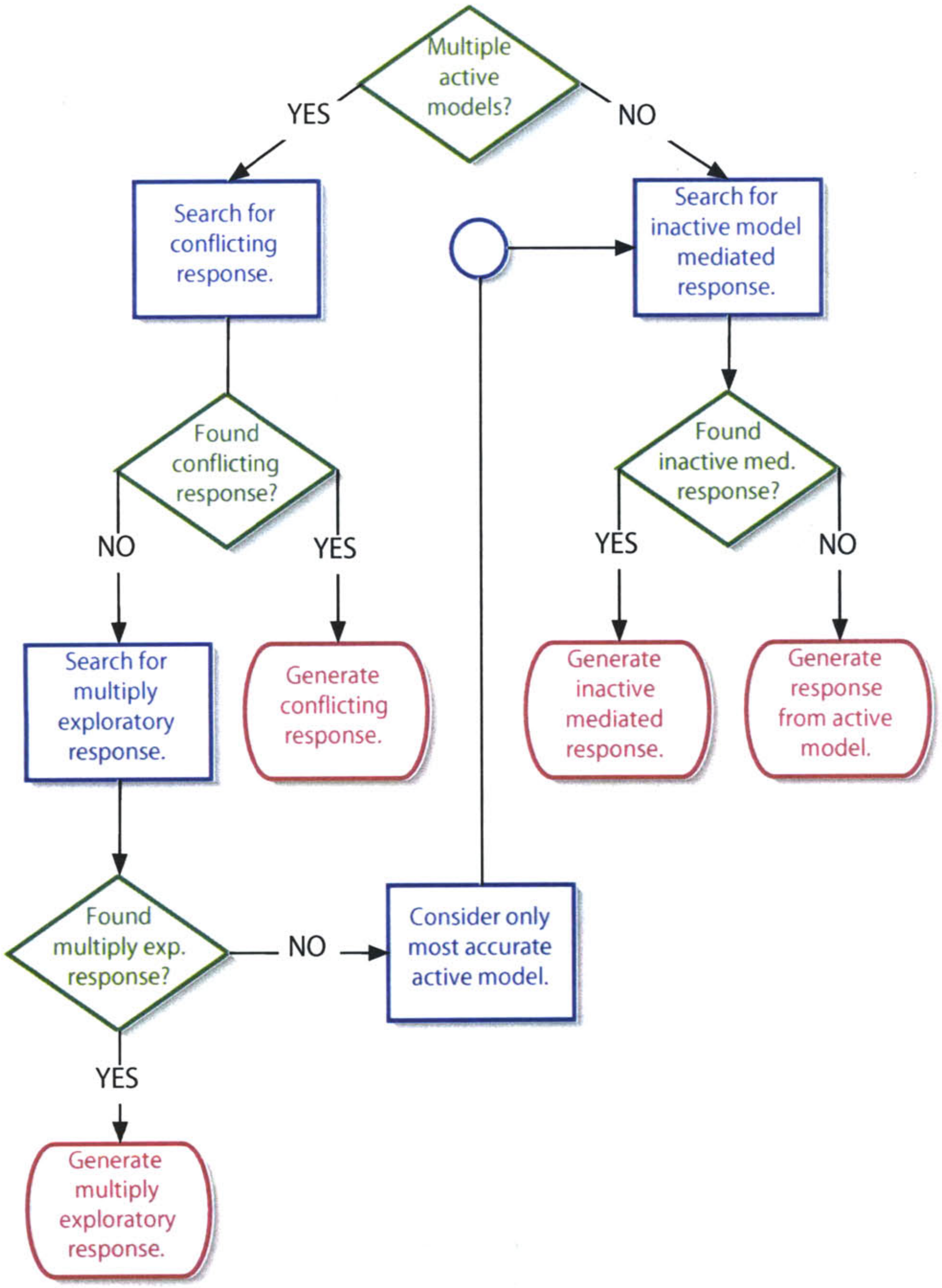

Multiple active models?
YES
NO
Search for conflicting response.
Search for inactive model mediated response.
Found conflicting response?
NO
YES
Found inactive med. response?
YES
NO
Search for multiply exploratory response.
Generate conflicting response.
Generate inactive mediated response.
Generate response from active model.
Found multiply exp. response?
NO
Consider only most accurate active model.
YES
Generate multiply exploratory response.

figure 5.8 Schematic of the logic governing the exploratory SLT. Conditional branch points are shown as green diamonds, operation points as blue rectangles, and output generation points as red rounded rectangles. Specific details for each of the operation and output generation points are given in the text.

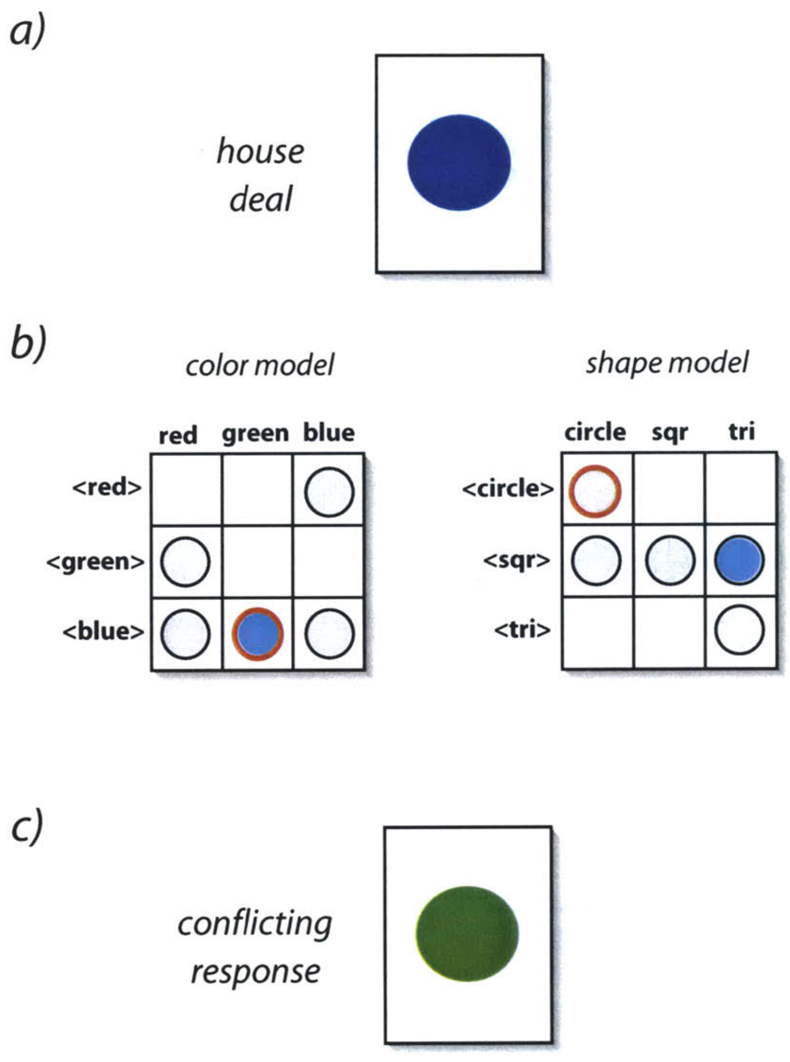


figure 5.9 An illustration of how multiple active models can make conflicting predictions for a given response card. Given the card dealt by the House (**a**), the content of the two active models (**b**) can be combined to generate a response card (**c**) for which the models make conflicting predictions. Specifically, as the red highlighted data points in (**b**) indicate, the color model predicts that a **green circle** response will lead to a match, while the shape model predicts that such a response will fail to match. See the accompanying text for more details.

sible partial hypotheses as to the matching rules: either **blue** is matched by **green**, or **square** is matched by **triangle**. Suppose that the House now deals a **blue circle** card (**figure 5.9a**). How should Hektor respond? Note that it is possible for the active models to make *conflicting predictions* for this card. While the shape model predicts that responding to a **circle** with a **circle** will lead to failure, the color model posits that responding to **blue** with **green** will lead to success (**figure 5.9b**); hence the models disagree on the expected outcome of responding with a **green circle** card (**figure 5.9c**). By responding with a **green circle**, therefore, Hektor is able to determine which of the two active models is in fact consistent with the current matching rule.

In this fashion, the exploratory SLT attempts to identify and exploit conflicts amongst the active models whenever possible. The strategy is a very powerful means of quickly narrowing down Hektor's hypothesis space, and is thus one of the most important contributors to the efficacy of the exploratory SLT and the matching toolbox generally.

Of course, it isn't always possible to coerce the active models into making conflicting predictions. If the exploratory SLT can't find a conflict for the House's card, it attempts to generate a **multiply exploratory response**. Such a response is one that will simultaneously expand the rule space coverage for two or more active models, as illustrated in **figure 5.10**.

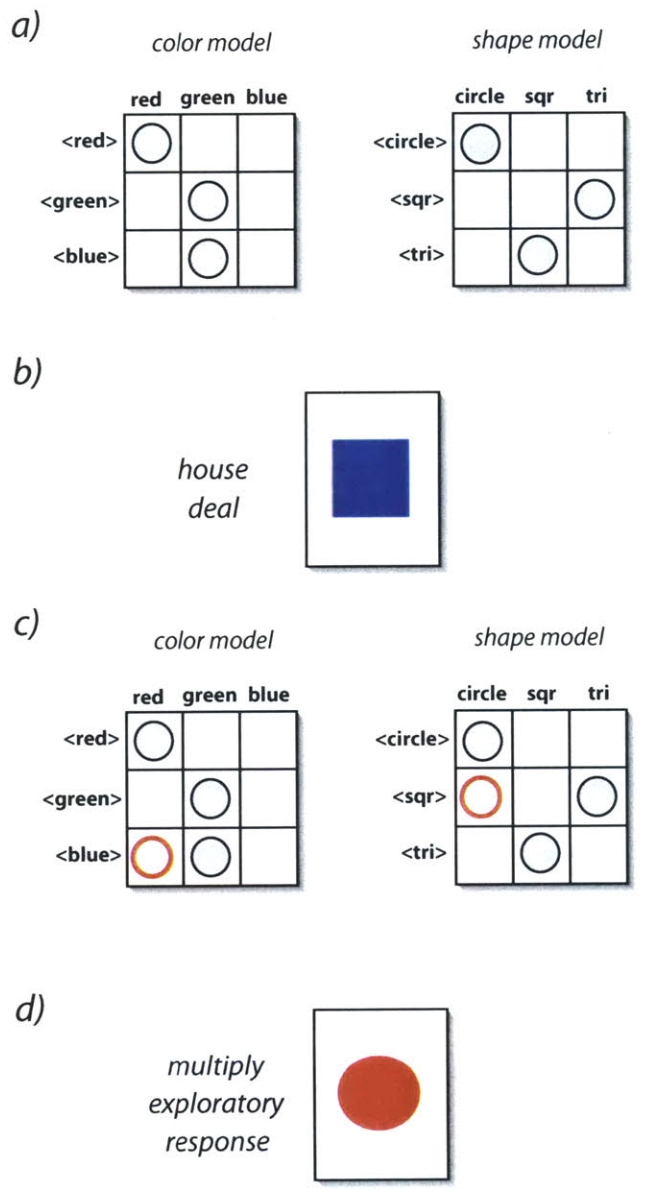


figure 5.10 An illustration of how a multiply exploratory response can simultaneously expand the rule space coverage of more than one active model. In (**c**), the red highlighting indicates the response to the House's deal that will succeed in expanding both models. See the accompanying text for more description.

In part **(a)** of the figure Hektor has again already played three hands, this time without discovering a match. Suppose that the House now deals a **blue square** card (**figure 5.10b**). By responding with a **red circle**, Hektor will simultaneously expand the rule space coverage for both of the active models (**figure 5.10c**). By generating this type of multiply exploratory response whenever possible, the exploratory SLT helps Hektor explore the matching rule space with maximum rapidity.

Finally, there can also be situations in which no conflicting nor multiply exploratory responses are possible. When the exploratory SLT encounters this dilemma it selects the active model with the most accurate prediction accuracy history and reverts to the single active model case, the logic for which we will now describe.

single active model

In the case of there being just a single active model, the exploratory SLTs mission is simply to expand that model's rule space coverage as quickly as possible. For any given House card then, the exploratory SLT attempts to produce a response card for which the active model has not yet recorded the result.

Recall, however, that a single active model can only specify *half* of a response card, since it only models one of the two possible card features, either shape or color. The exploratory SLT thus uses this situation as an opportunity to generate an **inactive model mediated response**. As the name implies, such a response is one that is formulated with input from an inactive matching model. Specifically, the card feature for which the active model has not provided a specification is determined by querying the inactive models for a potential match.

Consider **figure 5.11**. The figure depicts a situation in which the House began with a color-based matching rule for which Hektor was able to derive a correct matching model. The House then juggled the rules, causing the deactivation of the completed matching model as indicated by its lightened shading in **figure 5.11a**. Hektor is now attempting to learn the new rule set. Suppose the House now deals a **blue triangle** card (**figure 5.11b**). The active matching model specifies that the response card should be a square in order to expand its rule space coverage. To determine the *color* of the response card, the exploratory SLT now consults the inactive matching model. According to the inactive model, **blue** should be matched by **red** (**figure 5.11c**); Hektor thus plays a **red square** card in response (**figure 5.11d**).

Why do things this way? Recall that the purpose of storing inactive matching models is to give Hektor the ability to recognize when the House is re-using a previously encountered rule set (see **section 5.4.2**). Inactive model mediated responses *are the mechanism by which this recognition occurs.* That is, if the House reverts to a previously utilized rule set for which Hektor has an inactive matching model, that matching model needs

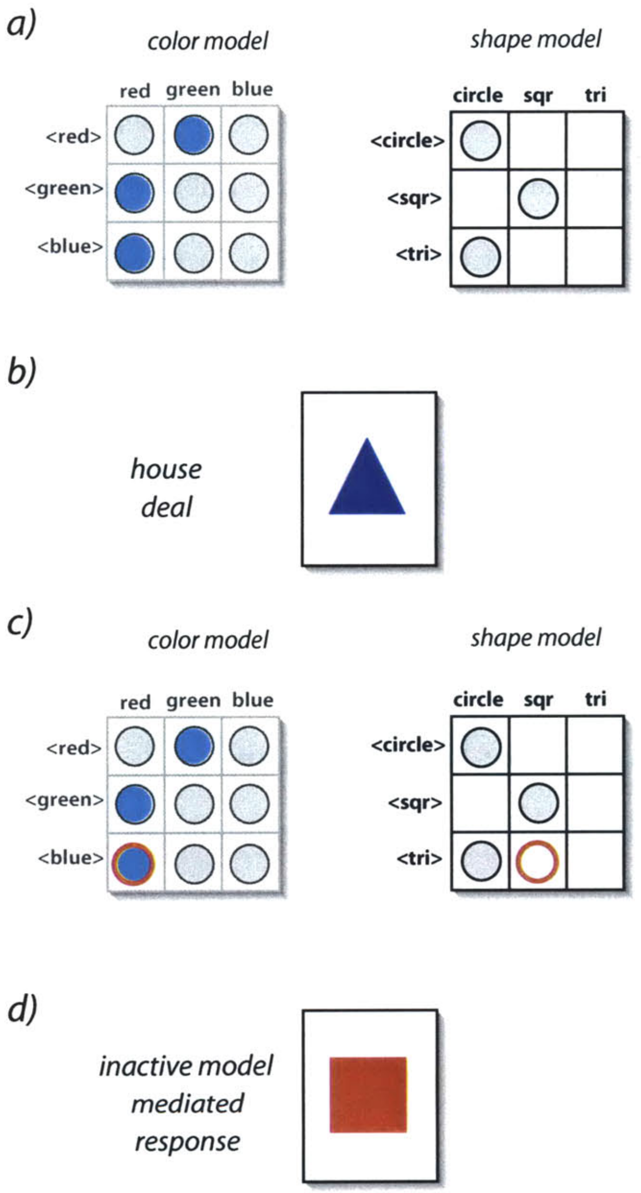


figure 5.11 An illustration of how an inactive model mediated response allows an inactive model (here indicated by lighter coloration) to exert control over Hektor's behavior. In **(c)**, observe that the inactive model combines its prediction that **red** should match **blue** with the active model's determination of the response card's shape. See the accompanying text for more detail.

an opportunity to prove its correspondence to the matching rules in order to be reactivated. By specifying a component of Hektor's responses in the manner described here, the inactive model can improve its prediction accuracy history and consequently become eligible for reactivation.

Note that the ability to generate an inactive model mediated response hinges on the availability of an inactive matching model whose dimension of interest is the opposite of that specified by the active matching model. That is, if we have an active shape model, we need an inactive color model to construct an inactive model mediated response. If this is not the case, the exploratory SLT will simply randomly select a value for the feature not specified by the active model.

5.6.3 phase two: taking action

Thus far we've examined the logic by which the exploratory SLT forms hypotheses regarding the best response card for a given deal. That information must now be communicated to the behavior system in order to take the appropriate action. This communication is accomplished by dispatching a behavioral parameter to the exploratory behavioral mode containing both propagated value and the identity of the desired response card. The exploratory behavioral mode is then activated by the propagated value, and ultimately causes Hektor to express the action of playing the specified card.

5.6.4 phase three: updating

Once the selected response card has been played, the exploratory SLT completes its learning epoch by passing the observed result to the matching toolbox where it is used to update the set of matching models as previously described (**section 5.4.2**).

5.7 the deployment and wager controller slts

Since the operations of the deployment and wager controller SLTs are quite tightly coupled, in this section I will consider the two systems together. I will first describe the conceptual role of both of the two SLTs, and then explain how their learning epochs interact in order to achieve their intended purposes.

5.7.1 responsibilities of the slts

Let's begin our consideration with the deployment SLT. The responsibility of the deployment SLT is to allow Hektor to mobilize his knowledge of the matching rules in order to achieve a desired goal. The goals that drive Hektor's knowledge deployment in the context of CheeseWhiz are either *i)* to win a particularly high-stakes hand, or *ii)* to deceive the House regarding the extent of his knowledge of the matching rules. The former goal is obviously useful when Hektor has complete (or nearly complete) knowledge of the matching rules and wishes to aggressively increase his cheese supply. The latter and rather more subtle goal is useful for buying time. Suppose for example that Hektor is converging on a complete model of the matching rules, but a recent string of wins threatens to prod the House into juggling the rule set. By intentionally losing a hand or two, Hektor can feign ignorance and perhaps gain more time to complete his matching rule hypothesis.

The wager controller SLT is responsible for managing Hektor's cheese supply and specifying the amount that Hektor should wager on each hand. Note that implicit in this responsibility is the fact that the wager controller SLT must determine whether or not the deployment SLT should be activated, and if so to what end. That is, if Hektor is running out of cheese, the wager controller SLT will want to both place a minimal bet *and* request an assured matching response from the deployment SLT. Let's consider how all of this takes place.

5.7.2 wager controller slt: activation conditions

As Hektor must place a wager on every hand, the wager controller SLT is continuously active. Prior to each House deal it executes a learning epoch as follows.

5.7.3 wager controller slt phase one: hypothesis formation

The hypotheses that the wager controller SLT forms can be thought of as having the form:

> Given the amount of cheese in the bank and the likelihood that the House will soon juggle the rules, the best wager for the current hand is ***x*** and the desired outcome is ***y***.

As has been previously noted (**section 5.3**), the current implementation of the wager controller SLT is a simplified one. This means that the three salient determinations underlying hypothesis formation, e.g. how likely the House is to juggle, the amount of cheese to wager, and the desired outcome of the hand, are all made according to a fixed (as opposed to adaptive) rule set. We'll return to this matter in our consideration of future work, where we'll make some suggestions as to how the wager controller SLT might be modified to become more powerful. In the meantime, let's consider the rule set that the wager controller SLT currently uses to drive its hypothesis formation.

A flow chart depicting the wager controller SLT's rule set is given in **figure 5.12**. As the figure shows, the first step in the wager controller SLTs operation is the assessment of whether Hektor should 'bet the farm' in order to radically increase his cheese supply or perhaps win the game entirely. This course of action will be appropriate when Hektor has a single matching model that is either complete or very nearly so, such that he is very likely to be able to successfully match any card that the House might deal. To make this determination, the SLT checks what percentage of the active matching model's rule space has been specified and compares

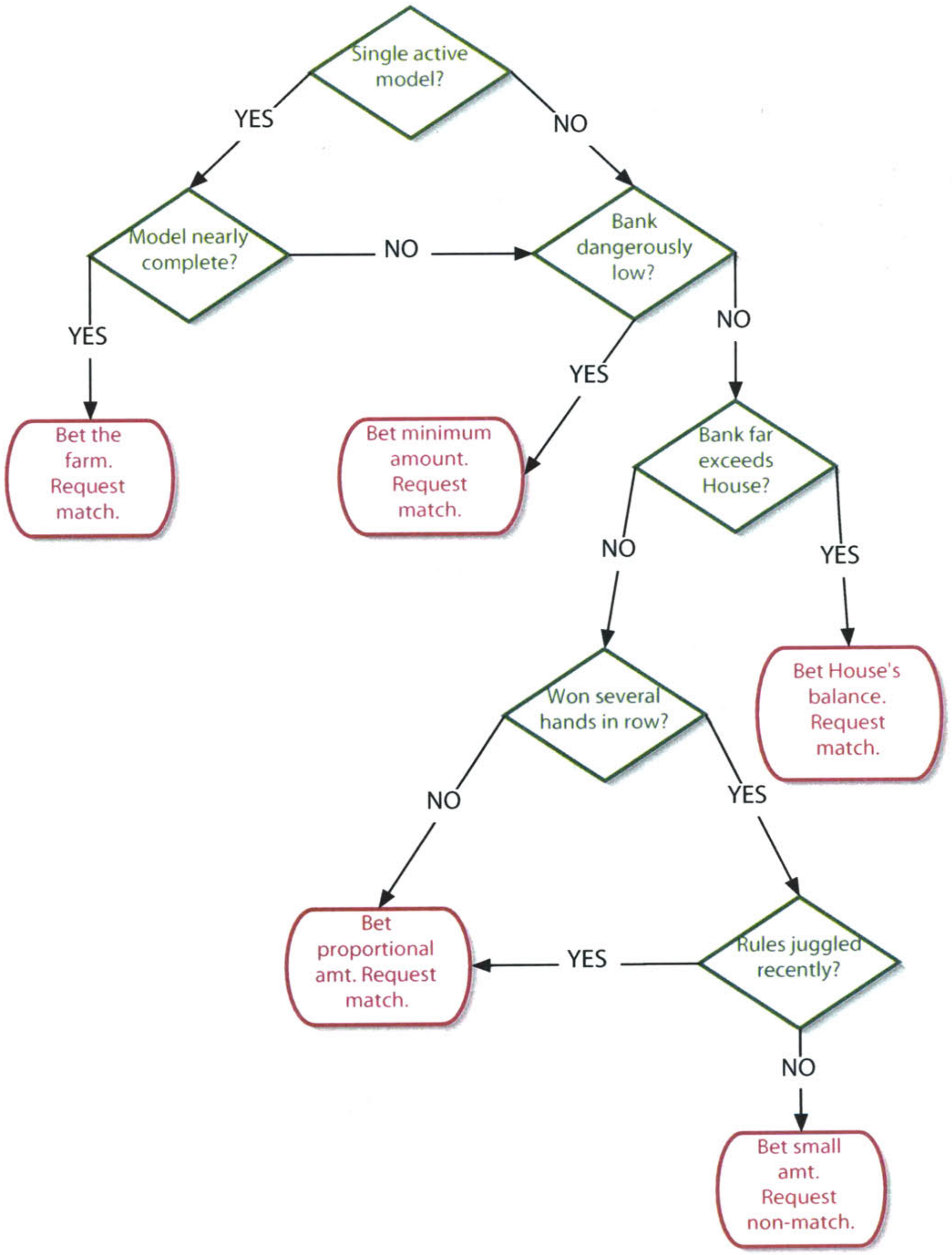

Single active model?
YES
NO
Model nearly complete?
NO
Bank dangerously low?
YES
Bet the farm. Request match.
YES
NO
Bet minimum amount. Request match.
Bank far exceeds House?
NO
YES
Bet House's balance. Request match.
Won several hands in row?
NO
YES
Bet proportional amt. Request match.
YES
Rules juggled recently?
NO
Bet small amt. Request non-match.

figure 5.12 A schematic illustration of the logic governing the wager controller SLT. Conditional branch points are shown as green diamonds and output generation points as red rounded rectangles. See the accompanying text for a detailed explanation of the figure's contents.

that value to a threshold.[3] If the percentage exceeds the threshold, the SLT specifies a wager equal to Hektor's total amount of cheese and designates matching as the desired outcome of the hand.

Of course, most of the time the wager controller SLT is not in a position to behave in such an aggressive fashion. If Hektor has multiple active models or a single model whose rule space is not yet sufficiently specified, the SLT will pursue a less risky betting strategy. Several distinct circumstances can pertain in this situation.

First, if Hektor's cheese supply has fallen below a caution threshold, the wager controller SLT will trigger very conservative behavior: it will specify the minimum bet of one piece of cheese and request a matching outcome for the current hand. Second, if Hektor is not dangerously low on cheese and his supply in fact exceeds that of the House by a specified multiplier (generally 1.5), the wager controller SLT will attempt to capitalize on the opportunity to win a game; it will specify a wager amount equal to the balance of the House's bank and again request a matching outcome. Third, if Hektor has neither a surfeit nor a deficit of cheese, the SLT will consider whether a little tactical deception might be in order. That is, if *i)* Hektor has won several hands in a row, *ii)* his active matching model is being refined but is not yet complete, and *iii)* the House has not recently juggled the rules, the wager controller SLT will specify a small wager and

3. Naturally this threshold is quite high, typically between 0.75 and 0.90.

request an intentional loss in order to deceive the House. As previously mentioned, the goal here is to prevent the House from juggling the rules, allowing more time for Hektor to complete his learning.

Finally, if none of the above special conditions holds, then the wager controller SLT will simply specify a wager that is proportional to the average percentage of the matching rule space covered by the active models. In this situation, no specification of a desired outcome is made.

5.7.4 wager controller slt phase two: taking action

The wager controller SLT is unique in that it takes action both through the behavior system and also through the superordinate deployment SLT. On the behavior system side, the wager controller SLT dispatches a behavioral parameter to the wagering motor primitive containing both propagated value and the amount of the desired wager. This causes Hektor to place the appropriate bet. On the SLT side, *if* the wager controller SLT arrived at a desired outcome for the current hand then a learning parameter containing that data is sent to the deployment SLT. This causes the deployment SLT to activate, which we will discuss momentarily.

5.7.5 wager controller slt phase three: updating

Since the wager controller SLT's rule set is currently static, the update phase of its learning epoch is empty. In future implementations this portion of the epoch would use the observed result of the completed hand to update the SLT's model of the House's behavior.

5.7.6 deployment slt: activation conditions

The deployment SLT is activated whenever it receives a learning parameter from the wager controller SLT specifying a desired outcome for the current hand. Upon activation the deployment SLT engages in a learning epoch as we shall now describe and then returns to its passive state.

5.7.7 deployment slt phase one: hypothesis formation

Hypotheses formed by the deployment SLT have the natural form:

> Playing card ***x*** will lead to desired outcome ***y***.

These hypotheses are generated in a straightforward manner by consulting the active matching models for a response appropriate to the desired outcome given the card that has been dealt by the House.

Of course, for a given House card and desired outcome, it will sometimes be the case that the active matching models are unable to suggest an appropriate response card. For example, suppose that Hektor has acquired twice as much cheese as the House, a situation that the wager controller SLT recognizes as a safe opportunity to attempt to win a game. The wager controller SLT will send a request to the deployment SLT for a matching response. Suppose further that Hektor's matching models are still in the process of being refined, such that they cannot specify a matching response for the card that the House deals. What response card should Hektor play in such a situation?

Since Hektor can't deploy his knowledge as desired in this situation, the next best thing is to use it as an opportunity to continue his exploration of the matching rule space. Accordingly, the deployment SLT will refrain from forming any hypothesis in this situation and instead *defer control to the exploratory SLT.* The exploratory SLT will then guide Hektor's response as previously described, in a fashion that maximizes the amount that will be learned from the outcome of the hand. This situation is a particularly elegant example of cooperation between SLTs.

5.7.8 deployment slt phase two: taking action

The deployment SLT takes action in much the same fashion as the exploratory SLT. A behavioral parameter containing propagated value and the desired response card is dispatched to the deployment behavioral mode, which in turn becomes activated and causes the specified card to be played.

5.7.9 deployment slt phase three: updating

After sending a behavioral parameter to the deployment behavioral mode the deployment SLT observes the outcome of the hand. If the outcome is what the deployment SLT had intended, then the current matching models are accurate and no adjustments need to be made. If, on the other hand, the outcome is not as expected, then the active matching models no longer correspond to the current matching rules. In this situation the deployment SLT sends a control signal to the matching toolbox causing the active models to be deactivated, new default models to be constructed, and (if appropriate) any recently accurate inactive models to be reactivated.

5.8 moving forward: evaluation

In this chapter we've developed a detailed understanding of Hektor's behavior and learning such that we're now in a position to begin considering their efficacy. In the next chapter I will undertake a thorough evaluation of Hektor's capabilities, assessing the strengths and weaknesses of the developmental design stance.

evaluation and discussion 6

In this chapter I undertake a thorough evaluation of Hektor and the developmental architecture on which he is based. Through both qualitative and quantitative analysis of Hektor's performance on the CheeseWhiz task, I will assess how successful this thesis' developmental approach to behavioral organization and learning has been. I will also consider some philosophical points regarding the relationship between this thesis and more traditional machine learning approaches and the ultimate scalability of developmental AI.

6.1 quantifying the learning task

Let's begin our evaluation by making some specific quantifications regarding the learning task Hektor is being asked to perform. First, recall that there are nine distinct cards in the CheeseWhiz game (see **figure 5.3**). Since the player is presumed to have an infinite supply of cards with which to respond to the House's deal, this means that there are *9 x 9 = 81* distinct deal-response combinations. Thus, in principle, the game's rule space contains 81 elements.

However, recall that Hektor's matching toolbox embeds the prior knowledge that matching rules must be specified either with respect to color or shape but not a combination of the two. This knowledge effectively constrains and divides the original 81 element rule space into two parallel *3 x 3 = 9* element rule spaces. For the purposes of evaluation then, I will take the overall size of the learning task's rule space to be 18 elements.

Given this quantification, approximately how many deals would be required to learn the matching rules via a simple exhaustive search strategy? A lower limit on this learning time is obviously 18 deals. However, note that this optimistic estimate presumes that the House will supply a sequence of 18 cards that can be used to perfectly span the rule space. In reality, since House cards are generated randomly, this circumstance is all but an impossibility; it is likely that exhaustive search would require significantly more than 18 deals to converge on a complete rule set. Nonetheless, for the purpose of maintaining a challenging standard of comparison, let's hold ourselves to the estimate of 18 deals as the benchmark we are attempting to better.

6.2 putting it all together: a detailed example

Having established an explicit standard of comparison, we are now in a position to begin judging the efficacy of Hektor's learning. Before we commence with a quantitative evaluation, however, it is perhaps helpful to get a more qualitative 'big picture' overview of how all the many components of Hektor's design fit together into an integrated whole. What kind of observable behavior, in other words, do all of the behavioral elements and specialized learning tools that have been described in the last two chapters give rise to? In this section I'll answer this question by considering a detailed 'transcript' from a real example of Hektor playing CheeseWhiz with a human user; I'll describe how Hektor's learning framework guides his behavior so as to converge on a model of the matching rules as efficiently as possible. Then, in the remainder of this chapter, I'll consider Hektor's average performance as quantified across numerous repetitions of individual trials like the one described here.

6.2.1 parameters for the example

In this example the user has initially specified a 'same color' matching rule in which color is the critical feature and like matches like. Given that there are 18 elements in the partitioned matching rule space, we'll opt for a challenge and say that the user is allowed to juggle the rules every nine deals.[1]

Note that under these circumstances, a naive strategy of exploring the state space by brute force would almost certainly fail.

6.2.2 a behavioral transcript

The behavioral transcript for this example is presented visually in **figure 6.1**. The following describes the figure's contents in detail.

getting your feet under you: observation

At the outset Hektor has no information about the matching rule other than his prior knowledge that either shape or color must be the critical feature. This prior knowledge is reflected in his initializing an empty matching model for each of these possible dimensions (**figure 6.1a**). Since the empty models provide no traction for engaging in active exploration of the rule space, Hektor's observational motivational subsystem becomes engaged. Hektor places the minimum bet of one piece of cheese, and responds to the House's opening **blue square** card with a randomly selected **red circle** card (**figure 6.1b**). This initial wager is unsuccessful, but both of Hektor's two active matching models now contain a data point (**figure 6.1c**).

sizing things up: exploration

Now that Hektor's matching models are non-empty, a control signal from the exploratory SLT causes Hektor's exploratory motivational subsystem

1. The question of how the interval between juggles impacts Hektor's success at CheeseWhiz is actually quite an interesting one; it will be considered in detail in **section 6.3.2**.

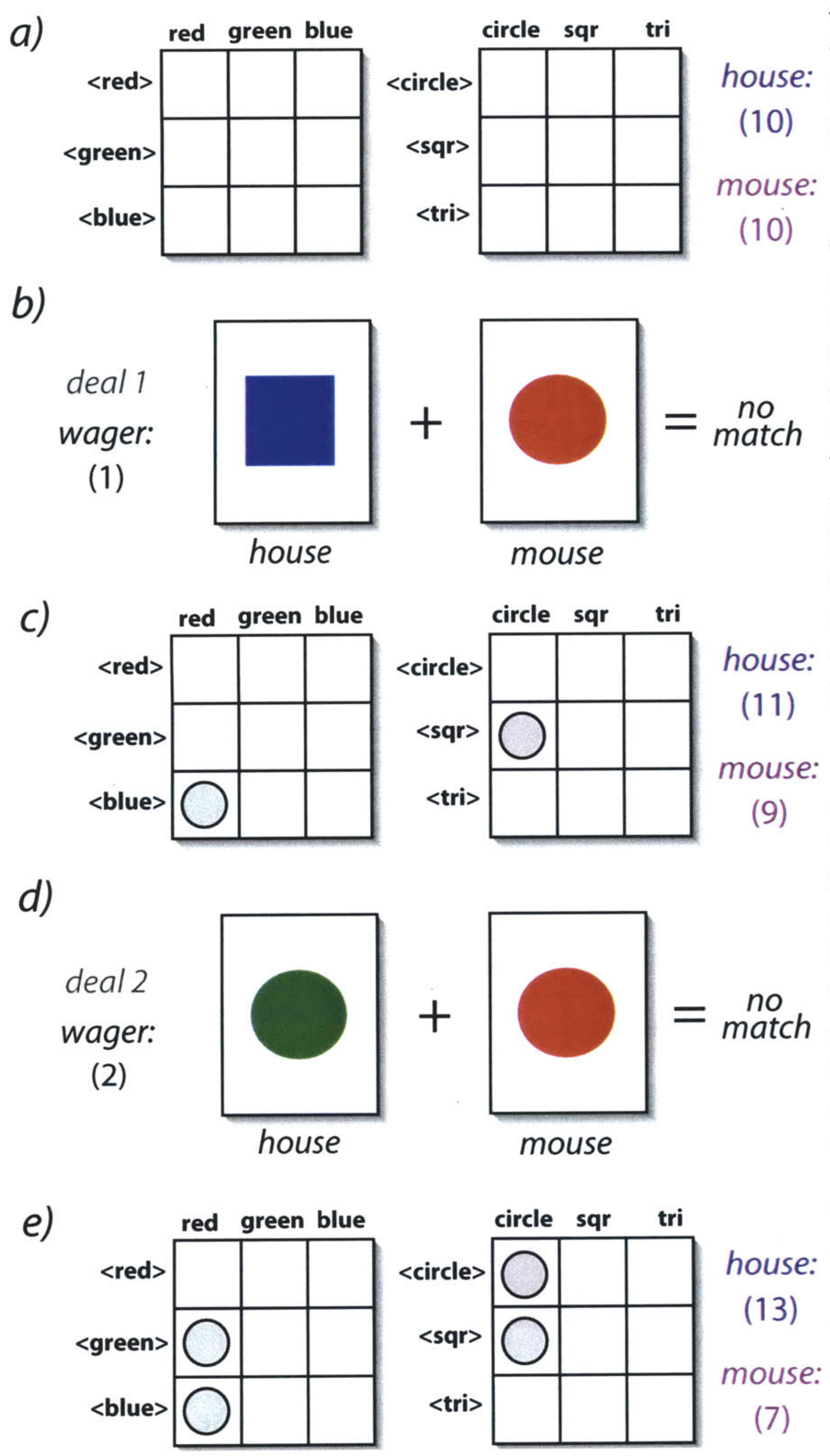
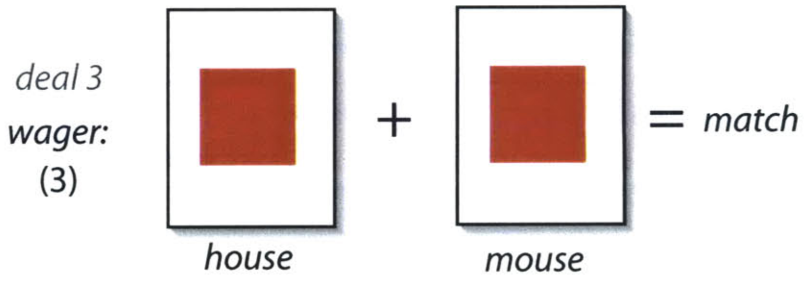


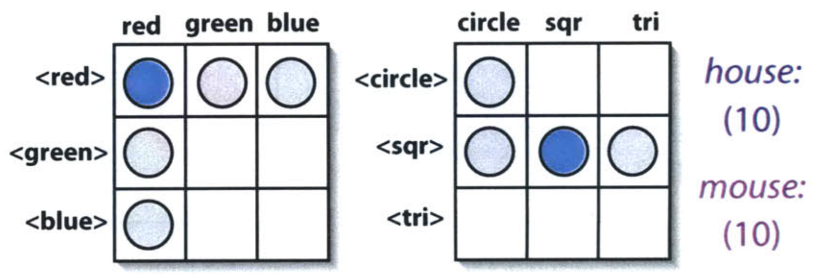
figure 6.1 A visual 'transcript' for a real instance of Hektor playing CheeseWhiz with the user. Matching models, depicted as grids, show the results of matching seed features (whose names are listed in angle-brackets) with response features. These results are represented as either a blank grid square (no data), a gray circle (no match), or a blue circle (match). Note that the changing balance of both Hektor's and the House's bank are given to the right of the depictions of the matching models. On each deal, the wager is indicated to the left of the cards. The cards themselves, labeled *house* or *mouse* as appropriate, show the deal-response combination that occurred on each hand; results are given to the right of the cards.

figure 6.1 (continued) Note that after deal 4, the shape matching model becomes inactive; this is indicated by its lighter shading in part *i* and thereafter. Also note that the House busts on deal 5, allowing Hektor to win his first game. After this win, in part *k*, observe that Hektor and the House have both had their banks reset to ten pieces of cheese for the start of the second game.

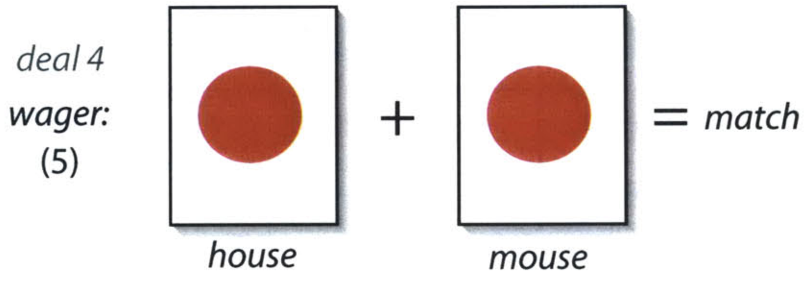
f)



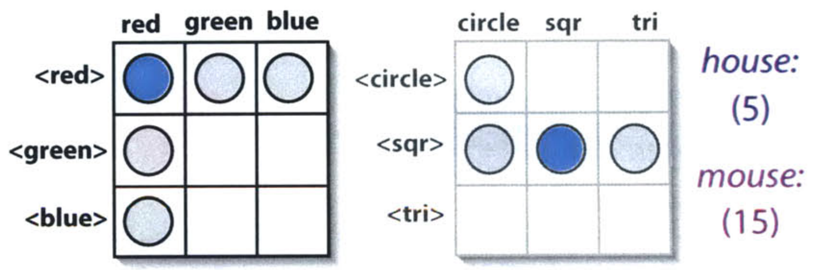
g)



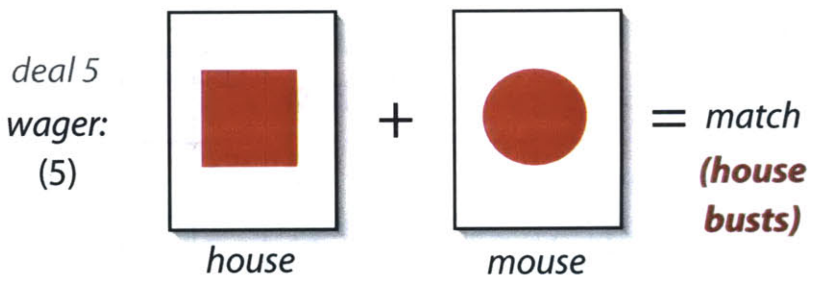
h)



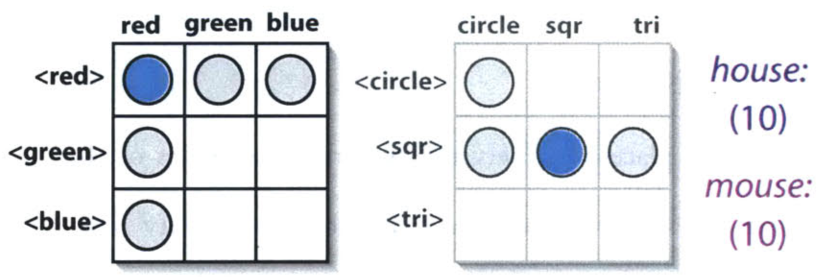
i)



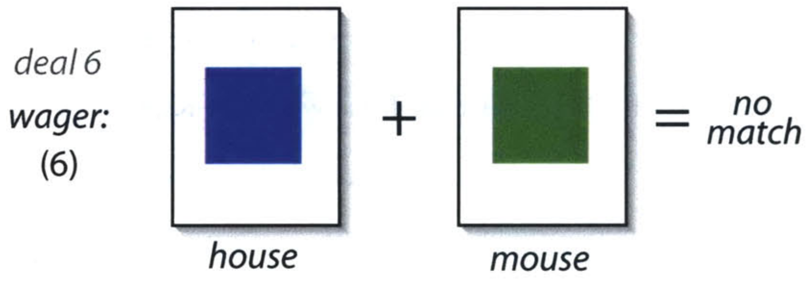
j)



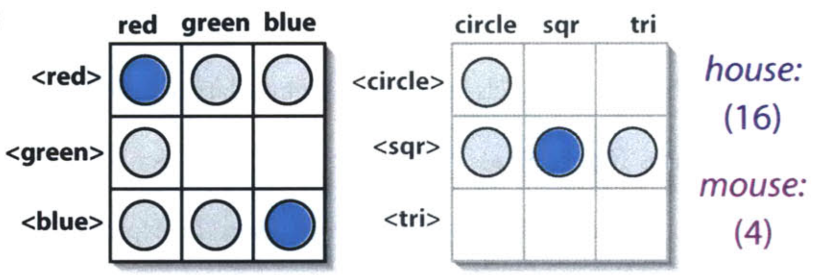
k)


red
green
blue
<red>
<green>
<blue>
circle
sqr
tri
<circle>
<sqr>
<tri>
house:
(10)
mouse:
(10)

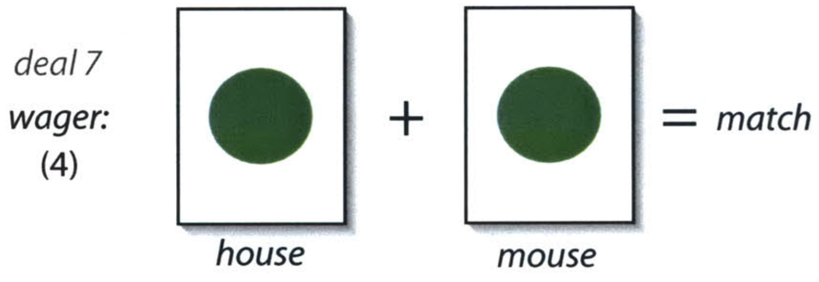
l)


deal 6
wager:
(6)
+
= no match
house
mouse

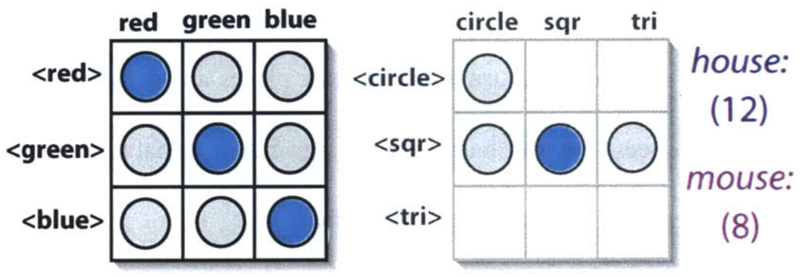
m)


red
green
blue
<red>
<green>
<blue>
circle
sqr
tri
<circle>
<sqr>
<tri>
house:
(16)
mouse:
(4)

n)


deal 7
wager:
(4)
+
= match
house
mouse

o)


red
green
blue
<red>
<green>
<blue>
circle
sqr
tri
<circle>
<sqr>
<tri>
house:
(12)
mouse:
(8)

to engage. Hektor's learning priority is now to differentiate the validity of the two active models.

Hektor places a conservative wager of two pieces of cheese, and the House plays a **green circle** card. Hektor's exploratory SLT attempts to determine the best response. It isn't possible to respond to this card in a way that might cause the two active models to diverge (e.g. the models don't yield conflicting predictions for any possible response to a **green circle**), so the SLT selects a response that will at least increase each model's coverage of its rule subspace; Hektor plays a **red circle** card, and is again unsuccessful **(figure 6.1d)**.

Since the active models now span slightly larger proportions of their rule subspaces **(figure 6.1e)**, Hektor increases his wager to three pieces of cheese. The House plays a **red square**. Once again the exploratory SLT can't identify a conflicting response for the two active models, so **red square** is played to expand rule subspace coverage. This time Hektor is successful **(figure 6.1f)**.

With success comes the ability to generalize; since each of the critical feature's values is matched by at most one response value, Hektor's matching toolbox is now able to significantly expand both model's coverage of their rule subspaces **(figure 6.1g)**. Hektor places an increased wager of five pieces of cheese on the basis of this significant expansion in his model's capabilities. The House plays a **red circle**.

Hektor's exploratory SLT now detects an opportunity. The color model predicts that responding to **red** with **red** will yield success, while the shape model predicts that responding to a **circle** with a **circle** will lead to failure (**figure 6.1g**). Hence, by responding with a **red circle** card, *Hektor will be able to determine which of the two currently active models is accurate.* Hektor plays a **red circle** and wins the wager (**figure 6.1h**). The shape model, having thus been proven inconsistent, is deactivated (**figure 6.1i**).

exploiting an opportunity: deployment

Hektor's deployment SLT is now activated by the fact that Hektor has significantly more cheese in his bank than the House. Though Hektor has not yet converged on a complete model of the matching rules, his deployment SLT recognizes that this is a good opportunity to play aggressively and attempt to win a game; learning, in other words, has temporarily taken a subsidiary role to the opportunistic goal of bankrupting the House. Hektor places a bet of five pieces of cheese, which if won would cause the House to bust. Unfortunately for the House, the next card in it's deck is a **red square**, which Hektor has already seen; Hektor responds with a **red circle** and takes the game (**figure 6.1j**).

completing the model: exploration

Though Hektor has won the first game, the House has only dealt five times and hence cannot yet juggle the rules. As the second game begins Hektor thus has a distinct advantage. As the House bank resets to ten pieces of cheese **(figure 6.1k)**. Hektor reverts to his exploratory motivational subsystem in order to continue refining his active matching model. Hektor wagers six pieces of cheese.

The House plays a **blue square**. This is good news for Hektor, as he can now finalize the **blue** portion of his matching model's rule subspace. Hektor responds with a **green square** and loses **(figure 6.1l)**, but his updated active model now covers seven of the nine possible elements of its rule subspace **(figure 6.1m)**.

the coup de grace: deployment

Since his active matching model now spans more than 75% of its rule subspace, Hektor shifts back to the deployment motivational subsystem; he is again going to play aggressively. Hektor bets all of his four remaining pieces of cheese. The House plays a **green circle**. Hektor's deployment SLT finds that it cannot locate a response to this card that is guaranteed to be successful **(figure 6.1m)**, and consequently defers to the exploratory SLT. The exploratory SLT suggests a **green circle** response; Hektor plays this card and wins the hand **(figure 6.1n)**. The active model generalizes, and Hektor has now converged on a complete, accurate model of the matching rules in just seven deals **(figure 6.1o)**. Note that the House still has to deal twice more before it can juggle the rules. Hektor places the maximum bet on each of these deals and wins two more games from the House. At the end of nine deals, Hektor has won three games without losing one.

6.2.3 preliminary discussion

The preceding is a good typical example of Hektor's ability to play CheeseWhiz effectively. Whereas converging on the correct matching rule using a naive exploration of the rule space would have likely required more than eighteen deals, Hektor has accomplished the task in just seven.

Several aspects of this performance deserve preliminary comment. First, note how Hektor transitions fluidly between his motivational subsystems, each corresponding to a context with differing behavioral and learning objectives. In particular, note how Hektor learns in the 'safe' contexts of observation and exploration on low-wager hands, and then deploys the results of that learning in the functional context of high-stakes hands. Hektor's transitions between behavioral contexts are not rigid and inflexible; for example, even though his rule model was far from complete at the outset of the fifth deal, Hektor still entered his deployment motivational subsystem in order to take advantage of the opportunity presented by the House's depleted cheese supply. True to the **first key lesson of development**, the intelligent management of behavioral contexts is central to Hektor's success.

Closely related to this first point, note also that cooperation between a distributed set of specialized learning tools allows Hektor to accomplish his learning objectives efficiently. Though each of the individual SLTs is specialized for learning within a particular behavioral context, they behave as an integrated whole under the supervision of the top-level matching toolbox. Hektor thus demonstrates that **the second key lesson of development**, that of learning using a multiplicity of domain-specific mechanisms, need not give rise to disjoint or ill-coordinated behavior.

With this detailed example in mind, let's now move on to a more quantitative analysis of Hektor's performance.

6.3 quantifying hektor's learning performance

In the following two studies I undertake a quantitative evaluation of Hektor's learning performance. Specifically, I assess and discuss both the average number of deals that Hektor requires to converge on a fixed rule set, and the minimum number of deals between juggles that are necessary for Hektor's success.

6.3.1 study one: average number of deals to convergence

The goal of this first study was to arrive at a robust estimate of the average number of deals that Hektor requires to converge on a fixed set of matching rules. By comparing this value to the size of the rule space, we will gain a more refined understanding of the efficiency of Hektor's learning mechanisms.

experimental design

This study consisted of fifty individual trials with randomly selected, fixed (e.g. non-juggled) matching rules. Within each trial, the House dealt randomly generated cards until Hektor had arrived at a complete and correct model of the matching rules. The number of deals required for this convergence to occur was recorded, as was the number of wins and losses (if any) that Hektor experienced along the way.

For this study, all of Hektor's free behavioral parameters were tuned towards maximal aggressiveness. In particular, Hektor was set to transition from passive observation to active exploration after having established just one element of his matching models' rule subspaces. Similarly, Hektor transitioned from exploration to deployment when his matching models had accounted for 75% of their rule subspaces. Hektor was also aggressive with respect to his wagering, always betting the maximum possible amount of cheese when in the deployment motivational subsystem. These settings were chosen to allow Hektor to converge on the matching rules as quickly as possible.

results and discussion

A histogram plotting the results of the fifty trials is shown in **figure 6.2**. Let's go over some of the datas' key features.

First and perhaps most importantly, the average number of deals required for Hektor to converge on a randomly selected set of matching

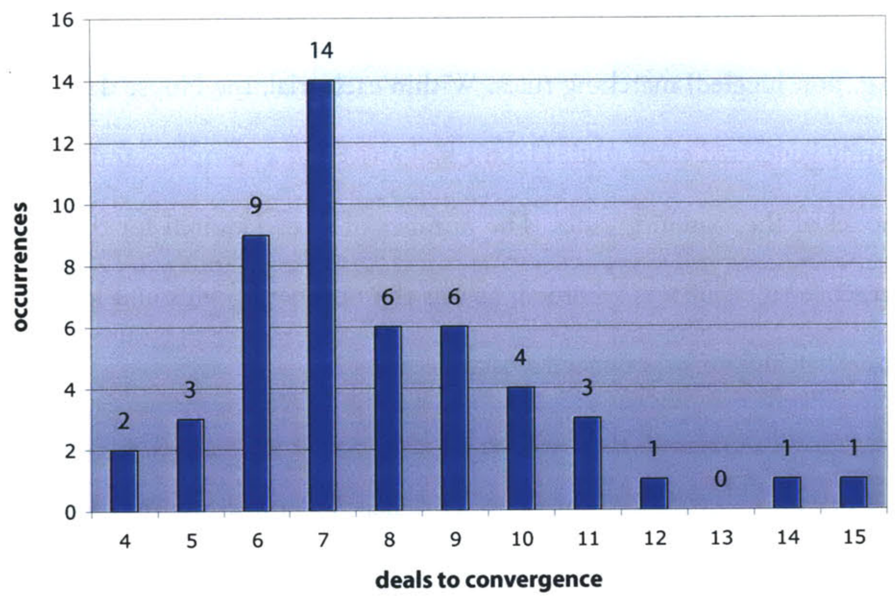


figure 6.2 Histogram depicting the distribution of fifty learning trials according to the number of deals required for convergence on a correct model of the matching rules. The mean number of deals for convergence is 7.8; the standard deviation is 2.3.

rules was found to be just 7.8, with a modest standard deviation of 2.3. In the best case, observed twice in the course of the fifty trials, Hektor was able to converge on the matching rules in just four deals. The maximum number of deals observed before convergence was 15.

The data shows that Hektor's learning strategy is significantly outpacing the performance that would be expected from a naive search of the rule space. Even in the worst case scenario of 14 deals, Hektor's performance was significantly better than the optimistic projection of 18 deals for exhaustive search. Moreover, the average requirement of just 7.8 deals to span a rule space with 18 elements is quite impressive.

An interesting and important observation is that the rapidity of Hektor's learning appears to be most notably constrained by his 'luck' with respect to being dealt helpful cards by the House. That is, in those cases where Hektor required a greater than average number of deals to converge on the matching rules, the difficulty was invariably that the House was slow in dealing the card(s) that were necessary for the rule space model to be completed. This fact is underscored by **figure 6.3**, which plots the number of wins experienced by Hektor as a function of the number of deals required for convergence. Though the data is somewhat noisy, particularly for the 12-15 deal trials for which one or fewer occurrences were observed, the overall trend is that a larger number of deals to convergence is correlated with a larger number of wins along the way. In other words, Hektor was frequently in the position of having the rule space 2/3 specified and simply having to wait for a card that would allow the final 1/3 of the rule space to be determined. While stuck in this waiting game, Hektor was able to accrue a large number of wins due to his knowledge of the subset of cards that were being dealt by the house. For the sake of comparison, a plot of Hektor's average number of losses as a function of the number of deals to convergence is given in **figure 6.4**.

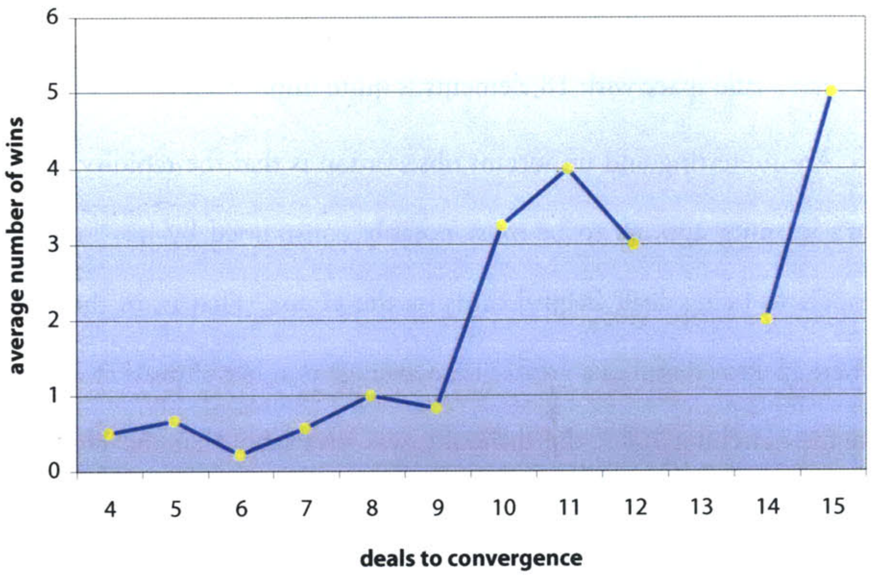


figure 6.3 A plot of the average number of wins encountered by Hektor while converging on fixed rule sets as a function of the number of deals required for convergence. Data is averaged over the course of fifty trials.

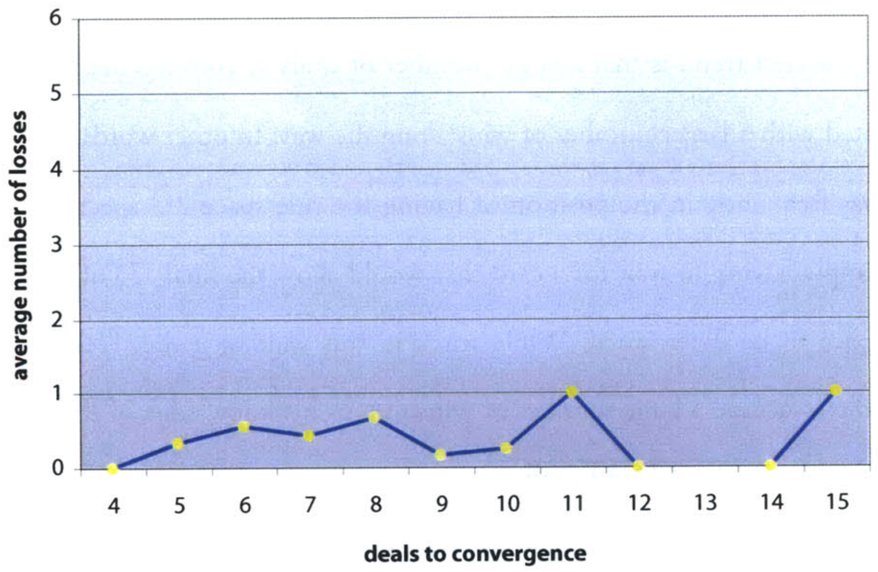


figure 6.4 A plot of the average number of losses encountered by Hektor while converging on fixed rule sets as a function of the number of deals required for convergence. Data is averaged over the course of fifty trials.

Comparing **figure 6.3** to **figure 6.4** makes it clear that Hektor is winning far more than he is losing while attempting to converge on a fixed rule set; the relevant averages are in fact 1.2 wins versus 0.4 losses per convergence. The natural next question is how this win:loss ratio might change when the House begins to juggle the matching rules. How well will Hektor's learning machinery cope with dynamically changing learning objectives? This question is the focus of study two.

6.3.2 study two: adaptability to changing rules

Having determined Hektor's mean number of deals to convergence for a stable rule set, the goal of this second study was to evaluate his ability to cope with varying rules. By varying the rules at different frequencies and observing Hektor's changing win:loss ratio, a profile of Hektor's ability to cope with the demands of changing learning objectives will be constructed.

experimental design

For the purpose of maintaining maximal comparability to **study one**, all of Hektor's behavioral parameters were kept constant in this study. Rule sets were randomized both at the beginning of each trial, and also whenever the minimum number of deals between juggles had elapsed. Juggle intervals of 5, 6, 7, 8, 9, and 10 deals were tested. For each juggle interval, two independent 100 deal-long trials were run. Hektor's total number of wins and losses on each trial were recorded.

results and discussion

Results from study two are summarized in **figure 6.5**, which plots the average number of wins and losses per 100 deal-long trial as a function of the juggle interval. Several important features of this data bear elaboration.

First, note that as expected there is a trend towards an increasing win:loss ratio as the number of deals between juggles increases. This point is more directly illustrated in **figure 6.6**, which plots the win:loss ratio as a function of juggle interval. As the House's ability to change the games' rules increases, winning consistently becomes an increasing challenge for Hektor.

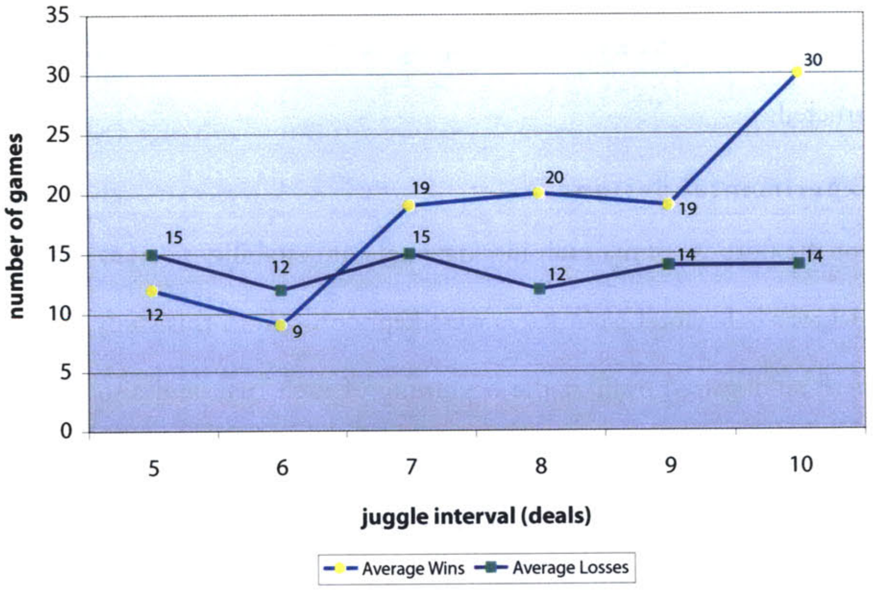


figure 6.5 Average number of wins and losses per 100 deals plotted against the interval between juggles. Data shown is averaged across two independent trials.

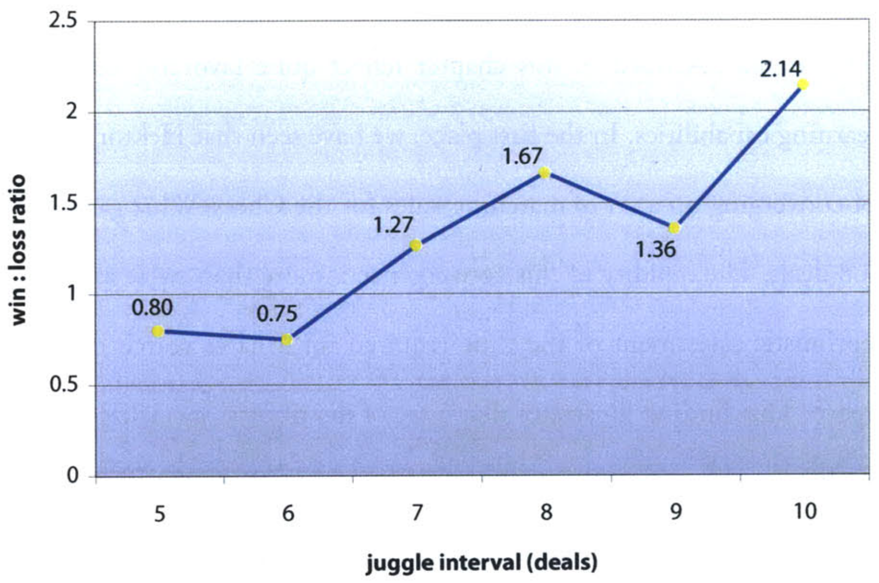


figure 6.6 Average win:loss ratio as a function of juggle interval, computed across two independent 100 deal trials.

More important than this unsurprising general trend, however, are the specific win:loss ratios observed. In particular, note that CheeseWhiz does not become unprofitable for Hektor until the House can randomize the rules ever six deals. This is quite an impressive performance. Even faced with an 18 element rule state space that is subject to randomization every seven trials, Hektor still manages to maintain a profitable win:loss ratio of 1.27. Taken together with the results of study one, these data demonstrate that Hektor is not only capable of learning rule sets quickly, but also of *re-learning* in an appropriately agile fashion when circumstances change.

6.4 overall discussion

The results described in this chapter reflect quite favorably on Hektor's learning capabilities. In the first place, we have seen that Hektor is capable of converging on a set of matching rules for the CheeseWhiz game in just 7.8 deals. The rapidity of this convergence is more than twice as fast as an optimistic assessment of the time required for a naive search of the rule space. This finding illustrates that a set of distributed specialized learning tools is capable of functioning in concert so as to efficiently achieve a complex learning objective. We have also seen that the rapidity of Hektor's learning is supported by the behavior system's ability to effectively deploy the knowledge he has gained. The developmental behavior architecture allows Hektor to transition smoothly between learning contexts and contexts in which learned behavior gives rise to its functional expression.

Secondly, we have seen that Hektor's architecture gives him the ability to manage his learning in an unpredictably varying environment. Like the biological systems from which we draw our conceptual inspiration, *Hektor learns precisely when he needs to learn.* Once Hektor has converged on a matching rule set, he disengages the more effortful aspects of his learning mechanism and focuses solely on the task of deploying the knowledge he has gained to greatest effect. When Hektor detects that the matching rule set must have changed, however, he is capable of immediately reverting to the observational and exploratory behavioral contexts and relearning as appropriate. This ability to move seamlessly between multiple behavioral contexts and multiple learning objectives is a key attribute underlying Hektor's ability to keep pace with a competitive human opponent.

6.4.1 hektor's scalability

What can be said about the scalability of Hektor's behavioral organization? There are at least two interesting questions to consider here. The first regards Hektor's ability to succeed at matching games with more than two features. How would we expect Hektor's performance to change if we were to play a matching game involving three, four or even more possible features?

Obviously more features would mean a larger overall rule space and consequently a slower average convergence time. However, there is nothing at all in Hektor's engineering that would be expected to scale poorly in this situation. Indeed, the exploratory SLTs ability to use conflicting predictions to arbitrate between multiple active models should scale extremely well with an increasing number of matching features; if anything, finding useful conflicts should become easier with more featural degrees of freedom. Hence, it is not expected that scaling to a greater number of matching features would pose a difficulty for Hektor.

A more challenging case of scaling might come from removing the restriction that matching rules must be expressed *within* features. What if matching rules could be formulated *across* features as well? The most significant impact of such a change would be the fact that it would no longer be possible to divide the matching problem into a set of parallel rule subspaces *a priori*; consequently, the overall rule space size would increase dramatically. Again, however, there is nothing fundamental in this situation that should prevent Hektor's current learning algorithms from being successful. In particular, the exploratory SLTs ability to maximize on conflicting model predictions should allow Hektor to efficiently distinguish between situations involving inter- and intra-featural matching rules, thereby rapidly whittling down the size of the rule subspace requiring thorough exploration.

6.4.2 difficulties and limitations

We've now seen that Hektor's behavior and learning architecture are quite successful overall with respect to the CheeseWhiz task. It is also instructive, however, to consider the difficulties and limitations that this chapter's experimentation has revealed. Though happily few in number, these points provide good fodder for considerations of future work.

a difficulty: recognizing previously encountered rule sets

In **section 5.4: the matching toolbox**, we described the mechanism by which inactive matching models corresponding to previously encountered rule sets are maintained in Hektor's memory. The motivating notion here was that by keeping these inactive models, Hektor would be able to quickly recognize when the House reverted to an old set of previously utilized matching rules. As discussed originally, however, the task of reactivating these old matching models at the appropriate time is a thornier logistical challenge than one might think. One can't simply reactivate an inactive model whenever its prediction accuracy history exceeds that of the currently active models, as this would means that new active models would be continually replaced by reactivated models before they could be refined. Recall that the proposed solution to this problem was to reactivate inactive models whose prediction accuracy history exceeded a specified threshold *only when all active models had been proven inconsistent.*

In the abstract this solution seems an ideal compromise. Since the exploratory SLT frequently references inactive models in order to determine the response card that Hektor should play (see **section 5.6**), inactive models should have the opportunity to help guide Hektor's pattern of responding, revealing situations in which they correspond to the matching rules more precisely than the active model. The only caveat, as originally mentioned in **section 5.4**, is that this will only work when it takes Hektor several deals to detect the inconsistency in his active model. The inactive models, in other words, must have time to prove their correspondence to the matching rules before the currently active model is discovered to be inconsistent. The experiments in this chapter have demonstrated that this situation seldom arises in practice. In fact, Hektor generally detects modifications of the matching rules on the first deal after they have occurred. In other words, it has been shown that the exploratory SLT doesn't 'accidentally' hit on the correct response because of input from inactive models as much as had been anticipated during the system's design. The mechanism for reactivating old models corresponding to previously encountered rule sets is consequently not as powerful as it might be.

a limitation: the assumption of one-shot inconsistency

According to the rules of CheeseWhiz, it is possible for Hektor to determine that a matching model is inconsistent after encountering just one unexpected result. When a matching model makes an incorrect prediction there is no question of noisy data or erroneous measurement – the model is simply wrong. While this assumption of **one-shot inconsistency** makes sense with respect to a card game, it is certainly true that many learning situations are not so obliging. In fact it is often important to be able to differentiate situations in which a new datum disagrees with an existing model because the model is incorrect from those situations in which the datum disagrees with the model because of noise in the sensor, etc. The assumption of one-shot inconsistency, therefore, is one that would be useful to remove from the matching toolbox in the future. As additional incentive, it should be noted that abandoning one-shot inconsistency should actually make the aforementioned challenge of detecting when to reactivate old matching models significantly easier. That is, if an active model's consistency were allowed to gradually decline over time, this would correspondingly give inactive models a more realistic opportunity to prove their correspondence to the matching rules. We will return to this topic in the next and final chapter's discussion of future work.

6.5 some philosophical evaluations

Thus far in this chapter we've evaluated Hektor, and the developmental approach to AI that he embodies, in both a qualitative and quantitative capacity. Before moving on to **chapter 7** and this thesis' ultimate conclusion, I would like to take a moment to consider some rather more philosophical points of evaluation.

6.5.1 machine learning revisited

Let's begin by revisiting the matter of traditional machine learning, first discussed in **section 3.2**. In this chapter there have been multiple mentions of how Hektor's performance on the CheeseWhiz task compares with the expected performance of an exhaustive search approach to the problem. While exhaustive search is useful as a basic reference point because of its conceptual tractability, it must be admitted that it sheds very little light on how more realistic alternative approaches to CheeseWhiz might compare to the developmental paradigm. How, in other words, might a competent student of traditional machine learning approach CheeseWhiz? What might the similarities and differences be relative to the developmental approach pursued in this thesis? While I will not distract from the main argument of this thesis by investigating these questions in an experimentally rigorous manner, I would like to consider them briefly in conceptual terms. In the following I will thus offer my suggestions as to how the developmental approach to learning adopted in this thesis might compare to that suggested by a more traditional machine learning perspective.

algorithmic similarities

With regard to the overall details of the learning algorithm (abstracted, for the moment, from its architectural context), I contend that a clever machine learning approach would probably differ very little from the overall algorithm embodied by Hektor's SLTs. After all, considered in isolation from their place in a larger behavior architecture, there is nothing explicitly developmental about the algorithmic details of the SLTs. The hypothesis formation mechanism of the exploratory SLT, for example, is really just an *ad hoc* machine learning algorithm, one that takes advantage of known constraints (e.g. the rules of CheeseWhiz) to maximize its efficiency and simplicity. While it would certainly be possible to apply a wide range of more sophisticated machine learning techniques to CheeseWhiz (reinforcement learning, neural networks, etc.),[2] the operations of Hektor's SLTs can be thought of as being abstractly equivalent to a kind of minimum complexity machine learning solution to the problem.

2. Such techniques are perhaps more properly termed 'over-sophisticated' in relation to the CheeseWhiz task.

This argument for continuity between the algorithmic details of Hektor's SLTs and traditional machine learning should come as no surprise given what was said in **section 3.2: a developmental take on machine learning**. Recall that my argument in that section was that machine learning and this thesis' developmental take on AI, despite the seeming distance between their respective mathematical formalism and biological inspiration, are actually deeply complementary. Specifically, I noted that the concept of the specialized learning tool could be understood as a kind of architectural 'wrapper' for domain-specific machine learning algorithms. Fundamentally then, what has been created in Hektor can be understood as a modest instance of machine learning: one that is embedded in the context of a novel behavioral architectural but not radically different in its abstract algorithmic detail from what a more traditional approach to the learning problem might yield.

architectural differences

Of course, I do not wish to argue that Hektor's learning paradigm is actually nothing more than traditional machine learning, so what is the critical distinction? The answer, as has been so conspicuously foreshadowed, is the fact of the learning algorithm's surrounding architectural context. Though algorithmically recognizable to a student of machine learning, Hektor's learning differs from more traditional approaches in that it takes place within, and is organized by, an explicitly developmental surrounding architecture. Leaving aside considerations of biological motivation for the moment, there are several core computational distinctions that this architecture imposes between itself and unembellished machine learning.

First, as we have discussed many times now, the developmental architecture insists strongly on the usage of domain-specific learning mechanisms. The more important consideration, however, is that *the developmental architecture sets the granularity of this domain-specificity in learning to correspond to the granularity of the creature's behavioral repertoire.* This is an important idea, essentially a statement of how the key computational lessons of development are implemented in this thesis' architecture, so let's take a moment to unpack it appropriately. What we are saying is that not only will learning tasks be decomposed using domain-specific SLTs, but each of these SLTs will exist at a level of specificity that allows it to correspond to a specific element in the behavior system. This quality of 'matching granularity' makes it possible for acquired knowledge to be repurposed almost effortlessly between distinct behavioral contexts. That is, to impose an AI slant on an example from Leyhausen [43], if a computational felid has learned how to pounce effectively in a playful context, then it will *automatically* know how to pounce effectively in the hunting context *because* that knowledge is maintained in an SLT that is *directly linked* to the pouncing motor primitive. The developmental architecture, in other words, uses the domain-specificity of SLTs to collapse as much as possible the distinction between behavior and learning; learning is envisioned as a consequence of behaving, not a parallel process.

This, I contend, is what constitutes the real distinction between the developmental approach to learning utilized in Hektor and what might be expected from a more traditional machine learning perspective. Hektor's learning is not only domain specific, but domain specific in a way that is carefully designed to map onto the elements of his behavioral repertoire. A traditional machine learning approach, with its emphasis on algorithm rather than architecture, would be unlikely to organize learning in this manner.

So why have I done things this way in this thesis? The motivation for this design comes from considerations of the ultimate scalability of the developmental architecture. It is my contention that as the sophistication of our computational creatures increases (e.g. as development becomes an increasingly good metaphor for the way in which these creatures change over time), this kind of architecturally imposed learning organization will scale more readily than traditional machine learning. That is, as creatures come to have more and more sophisticated behavioral capabilities, it will become increasingly important (indeed, necessary) to manage learning and behavior *together* rather than separately. The developmental approach detailed in this thesis accomplishes this by defining an explicit architectural specification for the relationship between learning and behavior – a specification that mirrors the relationship observed in the ontogeny of living animals.

6.5.2 on the long-term scalability of developmental ai

In the last section we hypothesized that the developmental learning organization employed in this thesis is likely to yield its most significant rewards as the complexity of our creatures increases. It is a truly developmental creature, in other words, that will best demonstrate the ultimate capabilities of the developmental architecture. Of course, this argument begs the question as to what developmental AIs prognosis for long-term scalability is. What are the challenges, architectural or otherwise, that will need to be overcome to progress from a developmentally inspired creature such as Hektor to a creature that is developmental in a more profound sense of the word?

This is a very difficult question to answer, and not one that can be approached with real certainty at this early stage. If I might be permitted a brief interlude for prognostication, however, I would begin by observing that there is always room to improve any computational architecture, especially one charged with a task as complex as managing the learning and behavior of an autonomous creature. The architecture defined in this thesis is certainly no exception to this rule. Indeed, in the future work section of the next chapter we will begin the consideration of several varieties of possible improvements that most immediately suggest themselves. That said, however, I would argue that constraints on this thesis' architecture itself are unlikely to be the most immediate stumbling block with regard to the realization of more fully developmental creatures. The general hierarchical design of the behavior system, inspired by a wealth of both ethological and computational theorizing ([1], [6], [24], [25], [70], [34], [69], [35], [71]), is exceedingly flexible and does not pose any obvious barriers with respect to supporting more elaborate behavioral repertoires. The architecture as it currently exists, for example, already has the ability to support multiple goals (in the form of multiple motivational subsystems or groups of such subsystems) and could easily be extended to support multiple simultaneous actions (using multiple threads of behavior selection and cross-exclusion between motor primitives). Rather, I would predict that the more significant barriers to the creation of full-fledged developmental creatures will cluster around issues of environmental complexity.

Development, I have argued, is largely about architecture, about how one organizes a learning system to adapt to the demands of highly sophisticated, complex, and dynamic environments. In order to fully apply these lessons, therefore, it is clear that we need to have environments that are realistically complex enough to make the requisite organizational overhead worthwhile. This, it turns out, is often quite a challenge at present.

As a purely anecdotal example, the Synthetic Characters research group (within which my research was situated) spent a great deal of time over the summer of 2004 brainstorming ideas for a research installation that would showcase the kinds of developmental insights discussed in this thesis. By far the greatest challenge in this process was simply that of conceiving of an environment in which there would be enough complexity to make the importance of development manifest. Without suitable environmental support (or, in this case, a suitable accompanying thesis!), developmental AI runs the risk of appearing to be little more than an interesting but perhaps over-complicated approach to behavioral organization.

This challenge of creating environments with enough complexity to fully showcase developmental organization is particularly acute with respect to the kind of wholly virtual environment used in this thesis. In a completely virtual world all complexity must be created by the designer, a process that is extremely labor intensive. Thus, I would argue that the creation of more sophisticated developmental creatures may well be better supported by the domains of robotics or mixed-reality environments (such as the ALIVE system of Maes *et al.* [44]). The real world is quite complex enough for anyone's purposes; by situating our developmental creatures within its complexity (or, initially, a partially attenuated version of its complexity) we can avoid the potentially infeasible burdens of hand-coding both complex creatures *and* complex worlds for them to inhabit. By challenging developmental AI creatures in more sophisticated environments, it will begin to be possible to more realistically assess the approaches potential for long-term scalability.

Having now evaluated Hektor and this thesis' developmental architecture from qualitative, quantitative, and even philosophical perspectives, let us turn our attention to the conclusion of the thesis. In the next and final chapter I will summarize the contributions that this thesis has made and make some suggestions regarding potentially profitable avenues of future research.

conclusions and future directions 7

In this chapter I conclude my thesis by reviewing its major contributions and offering some suggestions for future research.

7.1 summary of contributions

This thesis has covered a significant amount of intellectual ground, from a synthesis of the literature on animal cognitive development to a completed application of that synthesis to the challenge of computational learning. From this spectrum of research, I submit that the following contributions have been made:

1. **An elucidation of the computational significance of development.** From a broad review of the ethological literature on development, I have identi-

fied two key lessons with ready applicability to the design of computational systems.

First, I have demonstrated that during development the *when* of learning is often just as important as the *how*. This suggests that we need to pay close attention to the question of how our AI architectures will select and identify potentially valuable learning contexts. In particular, the development of animals from oscines to felids makes the point that the best context for learning a given skill will often be quite different from the context in which that skill will ultimately be expressed.

Second, I have shown that contrary to AIs prevailing desire for domain-general learning mechanisms, the process of development is often guided by a multiplicity of highly domain-specific cognitive tools. Often these tools endow developing animals with an innate "instinct to learn" [45] by embedding significant prior knowledge of their learning objectives. These ideas were used to motivate the proposal of **specialized learning tools**, a framework for approaching computational learning using a distributed network of domain-specific mechanisms.

2. The implementation of a novel behavioral and learning architecture motivated by the key computational lessons of development. The ideas drawn from the literature on development were used to design a developmental AI architecture. The architecture features a hierarchical, motivation-based behavior system consistent with the theoretical structures of the ethological behavior systems tradition ([1], [6], [24], [25], [70], [34], [69], [35], [71]). By structuring the behavior system's hierarchy as a lattice, behavioral elements can easily be shared between different parents; this allows behavioral modifications that are learned in one context to be flexibly repurposed into others. An intelligent behavior selection algorithm takes advantage of the system's hierarchical structure to preserve behavioral coherence when desirable while retaining the agility to switch priorities rapidly in response to unexpected changes in the environment.

Learning is accomplished using a set of specialized learning tools that may be distributed as needed throughout the behavior system. While individual SLTs are each tailored for a specific learning objective, they can be integrated into hierarchical **toolboxes** in order to cooperatively deconstruct complex learning tasks. Coordination between SLTs and the behavioral elements to which they are attached allows the learning process to be tightly integrated into the creature's overall behavior.

3. The successful deployment of the developmental architecture in a computational case study featuring a competitive learning interaction between an autonomous character and a human user. The developmental architecture implemented in this thesis was tested in a computational case study. The study involved Hektor the Mouse, who could attempt to win cheese by wagering with a human user on a card matching game called CheeseWhiz. Hektor's learning task was to quickly learn the user-selected matching rules that governed the game and to adjust appropriately when the user secretly altered them.

While constrained enough to be tractable, the CheeseWhiz learning task demanded the exploration of an 18 element rule space under significant time constraints and was therefore a non-trivial challenge. Using the hierarchical behavior system and a suite of specialized learning tools, however, Hektor was shown to be capable of converging on a correct model of the matching rules in an average of just 7.8 deals. Moreover, Hektor was able to maintain a highly favorable win:loss ratio of 1.5 even when the user had the ability to randomize the 18 element rule space every 7 deals.

The net effect of these contributions has been to establish a critical proof-of-concept for the developmental approach to AI proposed in this thesis. With this initial exploratory work accomplished, there is now a vast selection of exciting possibilities for future research. In the next section I will review some of the possibilities for enlarging on the developmental AI paradigm.

7.2 future research directions

What are some of the future challenges suggested by this work? While the potential impact of developmental design principles on AI is very broad, I'll focus here on the possibilities for future work that are most directly suggested by this thesis.

7.2.1 hektor-specific improvements

To begin with the most specific, there are several ways in which Hektor might be improved in the future.

improvements to matching models and their management

As was discussed in **chapter 6**, it would be worthwhile to modify the matching toolbox to support graded model inconsistency, thereby giving Hektor the ability to cope with potentially noisy data. One can think of many interesting modifications to the rules of CheeseWhiz that might make this ability valuable (for example, perhaps the House has the ability to periodically bluff a matching rule change for one or two deals); more generally, the ability to accommodate imperfect data is something that would obviously be extremely useful for applications of the matching toolbox outside the domain of CheeseWhiz.

A second improvement, closely related to that of supporting graded model inconsistency, would be to revamp the mechanism by which Hektor manages his inactive models. As previously mentioned, graded model inconsistency would actually go a significant distance towards improving matters in this area simply by giving the existing engineering the opportunity to function more effectively. Beyond this amelioration, however, there is room to consider alternative model reactivation strategies that could boost performance even further. One idea would be to use a pattern completion motif to detect when an inactive model should potentially be reactivated. That is, suppose the matching toolbox detects that an active matching model's specified rules are a subset of the rules for an inactive model, indicating that perhaps the House has reverted to an old rule set. In this situation, reactivating the inactive model in question could potentially shortcut the process of reconverging on the old rule set, thereby enabling Hektor to adapt very quickly. This is just one idea of course; the more central intuition is that making the model reactivation scheme more sophisticated would be quite likely to pay significant rewards in terms of enhanced performance.

improvements to hektor's slts

In **section 5.3** I discussed the fact that Hektor's observational and wager controller SLTs currently exist in a simplified form. While conforming to the architectural specifications for SLTs these learning tools do not at present actually *learn* anything – they are instead governed by static rule sets. It is quite likely that Hektor's performance on the CheeseWhiz task could be significantly improved by focusing some effort on the refinement of these components of his learning hierarchy.

The general idea for improving both of these SLTs would be to give them the ability to model and adapt to relevant aspects of the House's behavior. Rather than simply triggering Hektor to play a random card, for example, the observational SLT could condition its operation on a dynamically computed estimate of prior probabilities, preferentially playing cards that have been shown to be most likely to result in a match given the House's cumulative history of matching rule specifications. Similarly, the wager controller SLT could manage statistics regarding the House's likelihood of juggling the rules conditioned on some short history of preceding outcomes. Explicitly modeling the House's juggling behavior in this fashion would make for much more effective feigning than is possible using the current static rule set. Knowing when the House was likely to juggle the rules would also enable the wager controller SLT to manage Hektor's cheese supply more intelligently, adapting wager amounts to the likelihood that the current model of the matching rules has remained accurate.

While Hektor's ability to learn in the context of the CheeseWhiz task has already been found to be quite good, refining his simplified SLTs in this fashion should make him an even more shrewd competitor.

7.2.2 behavior system improvements

Having vetted the behavior system architecture through its application in Hektor, there are several candidates for future architectural improvement that suggest themselves. First, it would be worthwhile to consider how the message passing framework used to exchange information between behavioral elements might be made easier to use. While conceptually simple and powerful in its range of application, the work done for this thesis has revealed that the message passing paradigm can pose a surprising challenge for debugging. Thus, an obvious improvement would be to add a further layer of abstraction to the communication model implemented by behavioral elements, shielding users of the system from some of the nit-picky complications and easily avoided difficulties that commonly arise.

Second, one of the capabilities that was clearly suggested by the ethological literature but never fully elaborated in the behavior system is that of allowing for dynamically changing topologies, or large-scale reorganizations of parent-child relationships. Such reconfiguration is quite useful (conceptually at least) in more complex developmental scenarios in which radically changing environmental demands necessitate a concomitantly radical change in behavioral organization. While this idea of allowing for dynamic 're-wiring' was part of the basic design of behavioral elements, it was unnecessary for Hektor and thus never fully explored. It would be quite interesting to investigate the benefits that might be obtained by utilizing and refining this dynamic reconfiguration ability.

7.2.3 specialized learning tool improvements

What future improvements might we envision for specialized learning tools? Thus far it has been shown that coordinating the activity of multiple SLTs with toolboxes is a productive means of approaching complex, hierarchically structured learning tasks. The next step in complexity, therefore, might be to consider implementing the ability for toolboxes to communicate and cooperate with one another. This would extend the scale of the learning problems for which SLTs would be useful.

More generally, it is intriguing to consider the continuing development of the SLT concept at the content-specific level. To understand what is meant by this, note that the core engineering for SLTs has been intentionally left quite open-ended: the **hypothesize-act-update** cycle of SLT function can be mapped onto virtually any learning situation. It's possible then to consider the development of a kind of SLT library, a set of architecturally analogous learning tools, aimed at a range of adaptive objectives, that can be installed in new creatures on a 'plug-and-play' basis. Of course the vision of such a learning library would require many years (and a great deal of hard work) to accomplish, but it is interesting to consider the benefits that would inhere in the widespread adoption of a standardized SLT-like learning architecture.

7.3 the final word

In conclusion, this thesis has demonstrated that the subject of cognitive development is a potentially rich source of inspiration for the design of AI architectures. Millions of years of evolution have converged on the process of development as the best means of creating creatures that can adapt flexibly to the changing demands of unpredictable environments. Rather than spending a few million years more trying to do it differently, we would do well to take the lessons of development seriously in the design of our own creations.

references

[1] Arkin, R. C., Fujita, M., Takagi, T., & Hasegawa, R. (2003). An Ethological and Emotional Basis for Human-Robot Interaction. *Autonomous Systems, 42*(3-4).

[2] Baerends, G. (1976). On Drive, Conflict, and Instinct, and the Functional Organization of Behavior, *Perspectives on Brain Research* (*Vol. 45*).

[3] Balsam, P. D., & Silver, R. (1994). Behavioral Change as a Result of Experience: Toward Principles of Learning and Development. In J. A. Hogan & J. J. Bolhuis (Eds.), *Causal Mechanisms of Behavioural Development* (pp. 327-357). Cambridge, UK: Cambridge University Press.

[4] Berlin, M., Buchsbaum, D., Cochran, J., Downie, M., Lyons, D., & Blumberg, B. (2004). *Max T. Mouse: A "Teasible" Synthetic Character.* Research sketch submitted to ACM SIGGRAPH 2004, Los Angeles, CA.

[5] Berlin, M. R. (2003). *Predatory Sequence Learning for Synthetic Characters.* S.M. Thesis, Massachusetts Institute of Technology, Cambridge, MA.

[6] Blumberg, B. (1996). *Old Tricks, New Dogs: Ethology and Interactive Creatures*. Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, MA.

[7] Blumberg, B. (2002). D-Learning: What Dog Learning Tells Us About Building Characters that Can Learn. In G. Lakemeyer & B. Nobel (Eds.), *Exploring Artificial Intelligence in the New Millennium*. San Francisco: Morgan Kaufmann.

[8] Blumberg, B., Downie, M., Ivanov, Y., Berlin, M., Johnson, M. P., & Tomlinson, B. (2002). *Integrated Learning for Synthetic Characters*. Proceedings of ACM SIGGRAPH 2002, San Antonio, Texas.

[9] Blumberg, B., & Gaylean, T. A. (1995). *Multi-Level Direction of Autonomous Creatures for Real-Time Virtual Environments*. Proceedings of ACM SIGGRAPH 1995, Los Angeles, CA.

[10] Bolhuis, J. J., & Hogan, J. A. (Eds.). (1999). *The Development of Animal Behavior: A Reader*. Oxford: Blackwell Publishers.

[11] Breazeal, C. (2002). *Designing Sociable Robots*. Cambridge, MA: MIT Press.

[12] Breazeal, C. (2002). Regulation and Entrainment in Human-Robot Interaction. *International Journal of Robotics Research*, *12*(10-11), 883-902.

[13] Breazeal, C. (2003). Toward Sociable Robots. *Robotics and Autonomous Systems*, *42*, 167-175.

[14] Breazeal, C., Brooks, R. A., Gray, J., Hoffman, G., Kidd, C., Lee, H., Lieberman, J., Lockerd, A., & Mulanda, D. (2004). Humanoid Robots as Cooperative Partners for People. Submitted to *International Journal of Humanoid Robots*.

[15] Breazeal, C., Hoffman, G., & Lockerd, A. (2004). *Teaching and Working with Robots as a Collaboration*. Submitted to AAMAS 2004.

[16] Breazeal, C., & Scassellati, B. (2000). Infant-like Social Interactions Between a Robot and a Human Caretaker. *Adaptive Behavior, 8*(1), 49-74.

[17] Brooks, R. A. (1985). *A Robust Layered Control System for a Mobile Robot* (Artificial Intelligence Laboratory, A.I. Memo 864). Cambridge, MA: Massachusetts Institute of Technology.

[18] Brooks, R. A. (2002). *Flesh and Machines: How Robots Will Change Us*: Pantheon Books.

[19] Brooks, R. A., Breazeal, C., Irie, R., Kemp, C. C., Marjanovic, M., Scassellati, B., & Williamson, M. M. (1998). *Alternative Essences of Intelligence*. Proceedings of the 1998 International Joint Conference on Artificial Intelligence.

[20] Burke, R. (2001). *It's About Time: Temporal Representations for Synthetic Characters*. S.M. Thesis, Massachusetts Institute of Technology, Cambridge, MA.

[21] Burke, R., Isla, D., Downie, M., Ivanov, Y., & Blumberg, B. (2001). *Creature Smarts: The Art and Architecture of a Virtual Brain*. Proceedings of the 2001 Computer Game Developers Conference.

[22] Carey, S. (1985). *Conceptual Change in Childhood*. Cambridge, MA: MIT Press.

[23] Coppinger, R. P., & Smith, C. K. (1990). A Model for Understanding the Evolution of Mammalian Behavior. In H. Genoways (Ed.), *Current Mammology, Vol. 2* (pp. 33-74). New York: Plenum Press.

[24] Dawkins, R. (1976). Hierarchical Organization: A Candidate Principle for Ethology. In P. Bateson & R. A. Hinde (Eds.), *Growing Points in Ethology*. Cambridge, UK: Cambridge University Press.

[25] Davey, G. (1989). *Ecological Learning Theory*. London: Routledge, Inc.

[26] Drescher, G. L. (1991). *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. Cambridge, MA: MIT Press.

[27] Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern Classification (2nd ed.)*. New York: Wiley Interscience.

[28] Foner, L. N., & Maes, P. (1994). Paying Attention to What's Important: Using Focus of Attention to Improve Unsupervised Learning. In D. Cliff & P. Husbands & J. A. Meyer & S. W. Wilson (Eds.), *From Animals to Animats: Proceedings of the Third International Conference on the Simulation of Adaptive Behavior.* Cambridge, MA: MIT Press.

[29] Girard, M., & Maciejewski, A. A. (1985). Computational Modeling for the Computer Animation of Legged Figures. *Computer Graphics, 19*(3), 263-270.

[30] Hall, W. G., & Williams, C. (1999). Suckling Isn't Feeding, or Is It? A Search for Developmental Continuities. In J. J. Bolhuis & J. A. Hogan (Eds.), *The Development of Animal Behavior: A Reader.* Oxford: Blackwell Publishers.

[31] Hauser, M. D. (1996). *The Evolution of Communication.* Cambridge, MA: MIT Press.

[32] Hauser, M. D., & Spelke, E. (2004). Evolutionary and Developmental Foundations of Human Knowledge. In M. Gazzaniga & N. Logothetis (Eds.), *The Cognitive Neurosciences, III.* Cambridge, MA: MIT Press.

[33] Hinde, R. A. (1970). *Animal Behavior.* New York: McGraw-Hill.

[34] Hogan, J. A. (1994). Structure and Development of Behavior Systems. *Psychonomic Bulletin and Review, 1*(4), 439-450.

[35] Hogan, J. A. (2001). Development of Behavior Systems. In E. M. Blass (Ed.), *Handbook of Behavioral Neurobiology* (Vol. 13, pp. 229-279). New York: Kluwer Academic Publishers.

[36] Hraber, P. T., Jones, T., & Forrest, S. (1997). The Ecology of Echo. *Artificial Life, 3*, 165-190.

[37] Joshi, A., & Weng, J. (2003, July 20-24). *Autonomous Mental Development in High Dimensional State and Action Spaces.* Paper presented at the 2003 International Joint Conference on Neural Networks, Portland, OR.

[38] Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research, 4,* 237-285.

[39] Konishi, M. (1965). The role of auditory feedback in the control of vocalization in the white-crowned sparrow. *Condor, 66,* 85-102.

[40] Kruijt, J. P. (1999). Ontogeny of Social Behavior in Burmese Red Junglefowl (*Gallus gallus spadiceus*) (*excerpt*). In J. J. Bolhuis & J. A. Hogan (Eds.), *The Development of Animal Behavior: A Reader.* Oxford: Blackwell Publishers.

[41] Kuhl, P. K. (1999). Speech, Language, and the Brain: Innate Preparation for Learning. In M. D. Hauser & M. Konishi (Eds.), *The Design of Animal Communication.* Cambridge, MA: MIT Press.

[42] Lemke, J. L. (1996). Self-Organization and Psychological Theory. *Theory and Psychology, 6*(2), 352-356.

[43] Leyhausen, P. (1973). On the Function of the Relative Hierarchy of Moods (As Exemplified by the Phylogenetic and Ontogenic Development of Prey-Catching in Carnivores). In K. Lorenz & P. Leyhausen (Eds.), *Motivation of Human and Animal Behavior: An Ethological View.* New York: Van Nostrand Reinhold Company.

[44] Maes, P., Darrell, T., Blumberg, B., & Pentland, A. (1996). The ALIVE System: Wireless, Full-Body Interaction with Autonomous Agents. *The ACM Special Issue on Multimedia and Multisensory Virtual Worlds.*

[45] Marler, P. (1991). The Instinct to Learn. In S. Carey & R. Gelman (Eds.), *The Epigenesis of Mind: Essays on Biology and Cognition* (pp. 37-66). Hillsdale, NJ: Erlbaum.

[46] Marler, P. (1997). Three Models of Song Learning: Evidence from Behavior. *Journal of Neurobiology, 33*, 501-516.

[47] Marler, P. (1999). On Innateness: Are Sparrow Songs "Learned" or "Innate"? In M. D. Hauser & M. Konishi (Eds.), *The Design of Animal Communication* (pp. 293-318). Cambridge, Massachusetts: MIT Press.

[48] Marler, P., & Sherman, V. (1983). Song structure without auditory feedback: Emendations of the auditory template hypothesis. *Animal Behavior, 33*, 57-71.

[49] Marler, P., & Tamura, M. (1962). Culturally transmitted patterns of vocal behavior in sparrows. *Science, 146*, 1483-1486.

[50] McKenna, M., & Zeltzer, D. (1990). Dynamic Simulation of Autonomous Legged Motion. *Computer Graphics, 24*(4), 29-38.

[51] Miller, D. B. (1997). The Effects of Nonobvious Forms of Experience on the Development of Instinctive Behavior. In C. Dent-Read & P. Zukow-Goldring (Eds.), *Evolving Explanations of Development.* Washington, D.C.: American Psychological Association.

[52] Minsky, M. (1985). *The Society of Mind.* New York: Simon and Schuster, Inc.

[53] Minsky, M. (2002). *The Emotion Machine.* Pre-publication draft.

[54] Mitchell, T. M. (1997). *Machine Learning.* Boston: WCB McGraw-Hill.

[55] Perlin, K., & Goldberg, A. (1996). *Improv: A System for Scripting Interactive Actors in Virtual Worlds.* Proceedings of ACM SIGGRAPH 1996, New Orleans, LA.

[56] Prince, C. G. (2002). *Introduction: The Second International Workshop on Epigenetic Robotics.* Proceedings of the Second International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems, Edinburgh, Scotland.

[57] Pryor, K. (1999). *Clicker Training for Dogs*. Waltham, MA: Sunshine Books.

[58] Ray, T. S. (1998). Selecting Naturally for Differentiation: Preliminary Evolutionary Results. *Complexity, 3*(5), 25-33.

[59] Reynolds, C. W. (1987). *Flocks, Herds, and Schools: A Distributed Behavioral Model.* Proceedings of ACM SIGGRAPH 1987, Anaheim, CA.

[60] Scassellati, B. (2000, September). *Theory of Mind for a Humanoid Robot.* Paper presented at the First IEEE/RSJ International Conference on Humanoid Robotics.

[61] Scassellati, B. (2003, July 20-24). *Investigating Models of Social Development Using a Humanoid Robot.* Paper presented at the 2003 International Joint Conference on Neural Networks, Portland, OR.

[62] Searle, J. R. (1998). How to Study Consciousness Scientifically. *Brain Research Reviews, 26*, 379-387.

[63] Shettleworth, S. (1994). The Varieties of Learning in Development: Toward a Common Framework. In J. A. Hogan & J. J. Bolhuis (Eds.), *Causal Mechanisms of Behavioural Development* (pp. 358-376). Cambridge, UK: Cambridge University Press.

[64] Sims, K. (1994, July 1994). *Evolving Virtual Creatures.* Proceedings of ACM SIGGRAPH 1994, Orlando, FL.

[65] Spelke, E. (2000). Core Knowledge. *American Psychologist, 55*, 1233-1243.

[66] Spelke, E. S. (2003). What Makes Us Smart? Core Knowledge and Natural Language. In D. Gentner & S. Goldin-Meadow (Eds.), *Language in Mind: Advances in the Study of Language and Thought.* Cambridge, MA: MIT Press.

[67] Sutton, R. R., & Barto, A. G. (2000). *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press.

[68] Thelen, E., & Smith, L. B. (1994). *A Dynamic Systems Approach to the Development of Cognition and Action.* Cambridge, MA: MIT Press.

[69] Timberlake, W. (2000). Motivational Modes in Behavior Systems. In R. R. Mowrer & S. B. Klein (Eds.), *Handbook of Contemporary Learning Theories* (pp. 155-209). Hillsdale, NJ: Erlbaum Associates.

[70] Timberlake, W., & Lucas, G. A. (1989). Behavior Systems and Learning: From Misbehavior to General Principles. In S. B. Klein & R. R. Mowrer (Eds.), *Contemporary Learning Theories: Instrumental Conditioning Theory and the Impact of Biological Constraints on Learning* (pp. 237-275). Hillsdale, NJ: Erlbaum Associates.

[71] Tinbergen, N. (1951). *The Study of Instinct.* New York: Oxford University Press.

[72] Tu, X., & Terzopoulos, D. (1994). *Artificial Fishes: Physics, Locomotion, Perception, Behavior.* Proceedings of ACM SIGGRAPH 1994, Orlando, FL.

[73] Waddington, C. H. (1999). Principles of Development and Differentiation (*excerpt*). In J. J. Bolhuis & J. A. Hogan (Eds.), *The Development of Animal Behavior: A Reader.* Oxford: Blackwell Publishers.

[74] Watkins, C. J., & Dayan, P. (1992). Q-Learning. *Machine Learning, 8.*

[75] Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M., & Thelen, E. (2001). Autonomous Mental Development by Robots and Animals. *Science, 291*, 599-600.

[76] Weng, J., Zhang, Y., & Chen, Y. (2003, July 20-24). *Developing Early Senses About the World: "Object Permanence" and Visuoauditory Real-time Learning.* Paper presented at the 2003 International Joint Conference on Neural Networks, Portland, OR.

[77] Whaling, C. S., Soha, J. A., Nelson, D. A., Lasley, B., & Marler, P. (1998). Photoperiod and tutor access affect the process of vocal learning. *Animal Behavior, 56*, 1075-1082.