Abstraction, Imagery, and Control in Cognitive Architecture

by

Samuel B. Wintermute

**A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2010**

Doctoral Committee:

**Professor John E. Laird, Chair
Professor Satinder Singh Baveja
Professor Benjamin Kuipers
Professor Richard L. Lewis**

For my parents.

Acknowledgements

First, I would like to thank my advisor, John Laird. I came to graduate school as a Master's student with no AI background, and right from the beginning John provided me with his advice and knowledge, and, most importantly, a foothold to get started in research. Since then, John's ability to help me when necessary and to let me go my own way at other times has made this experience much better than it could have been. The other members of my committee, Satinder Singh, Ben Kuipers, and Rick Lewis, have also all been very responsive throughout this process, and have all provided me with excellent feedback relevant to their areas of expertise, from which the work here has greatly benefited.

I would also like to thank the other current and former members of the Soar group at U of M, who have provided me with constant valuable feedback on my work, or at least have put up with it when I needed to talk something over out loud: Nate Derbinsky, Nick Gorski, Scott Lathrop, Bob Marinier, Andrew Nuxoll, Yongjia Wang, Joseph Xu, and everyone else who has sat in that office over the last five years. In particular, Joseph has been indispensable as a friend and collaborator, and Scott tolerated my constant questioning as I was attempting to build on his work just as he was leaving.

Finally, I would like to thank my family, along with my friends and housemates, who have supported me throughout graduate school.

Table of Contents

Dedication ii

Acknowledgements iii

List of Figures vii

List of Tables ix

Abstract x

Chapter I - Introduction 1

Chapter II - Motivation 7

2.1 Using Symbolic Representation for Decision-Making 7

2.2 Motivating Tasks 8

2.2.1 The Blocks World 9

2.2.2 The Pedestal Blocks World 11

2.2.3 Motion Planning for a Nonholonomic Car 12

2.3 Meta-Problems in Architecture Design 15

2.4 Imagery for Spatial Tasks 17

Chapter III - A Theoretical Architecture for Spatial Problems 20

3.1 Theory Description 20

3.2 High-level Claims and Evaluation Approach 26

3.3 Benefits of the Theory 27

Chapter IV - The Soar/SVS architecture 39

4.1 Overview 40

4.2 Perceptual Pointers 43

4.3 Spatial Scene Encoding 46

4.4 Predicate Extraction 48
4.5 Imagery 51
4.5.1 Predicate Projection 52
4.5.2 Memory Retrieval and Image Composition 54
4.5.3 Motion Processing 54
4.6 Aspects of the Theory in Soar/SVS 57
Chapter V - Reinforcement Learning Agents in Soar/SVS 60
5.1 State Abstraction and Imagery in Reinforcement Learning 60
5.1.1 The Pedestal Blocks World 62
5.1.2 Perceptual Abstraction in Pedestal Blocks World 63
5.2 State Abstraction, Action Modeling, and Imagery in Reinforcement Learning 66
5.2.1 Correctness in ReLAI 69
5.2.2 ReLAI and Perceptual Abstraction 73
5.3 ReLAI in Complex Problems 76
5.3.1 Frogger II Agent 78
5.3.2 Space Invaders Agent 83
5.3.3 Fast Eddie Agent 87
5.3.4 Summary of Video Game Experiments 91
Chapter VI – Motion Planning in Soar/SVS 94
6.1 The RRT Algorithm 95
6.2 RRT in Soar/SVS 97
6.3 Symbolic Soar Processing in RRT 100
6.4 Perceptual Abstraction and Irreducibility in Motion Planning 102
Chapter VII - Related Work 105
7.1 Qualitative Spatial Reasoning 105
7.2 Cognitive Architectures 106

7.2.1 Other Soar Extensions 106
7.2.2 ACT-R 109
7.3 Robotic Systems 111
7.4 Reinforcement Learning 113
7.5 Grounded Cognition 115
Chapter VIII - Discussion and Conclusion 118
8.1 Claims Revisited 118
8.2 Perceptual Abstraction, Irreducibility, and the Imagery Debate 125
8.3 Soar/SVS as a Cognitive Model 128
8.4 Contributions 130
8.5 Future Work 133
8.6 Conclusion 135
Appendix: Quantities and Symbolic Representation 136
References 144

List of Figures

Figure 1: A simple blocks world task. 9
Figure 2: A pedestal blocks world problem and its optimal solution state. 11
Figure 3: A situation in pedestal blocks world where nonlocal interactions are important if the agent is considering moving block C to pedestal 2. 12
Figure 4: A nonholonomic car motion planning problem. 14
Figure 5: Imagery in pedestal blocks world. 18
Figure 6: A basic non-imagery architecture (left) and an imagery architecture (right). .. 20
Figure 7: Alternative architecture designs lacking aspects of the imagery theory. 24
Figure 8: SVS System design.. 40
Figure 9: Top, blocks world information in Working Memory, Perceptual LTM, and Spatial Scene. Bottom, equivalent text representation of the extract-predicate structure. 45
Figure 10: An example of the symbolic encoding of the spatial scene in Soar. 47
Figure 11: Information derivable through spatial predicate extraction. 49
Figure 12: A Pedestal Blocks World task instance and its optimal solution. 62
Figure 13: Results of learning in pedestal blocks world showing advantage of imagery-augmented state abstraction. 65
Figure 14: The ReLAI algorithm as instantiated in Soar/SVS. 68
Figure 15: Results of learning in pedestal blocks world showing advantage of ReLAI. 69
Figure 16: Perceptual information in the game Frogger II, including object labels. 78
Figure 17: Performance of ReLAI vs. direct state abstraction in Frogger II. 82
Figure 18: Perceptual information in the game Space Invaders, including object labels. 83
Figure 19: Performance of ReLAI vs. direct state abstraction in Space Invaders. 86
Figure 20: Perceptual information in the game Fast Eddie, including object labels. 87
Figure 21: Performance of ReLAI vs. direct state abstraction in Fast Eddie. 91

Figure 22: A nonholonomic car motion planning problem. 94
Figure 23: The RRT Algorithm. 96
Figure 24: An example of RRT applied to car motion planning. 96
Figure 25: States of SVS Spatial Scene during RRT planning. 99
Figure 26: A trace of a Soar/SVS agent executing RRT planning. 100

List of Tables

Table 1: Aspects of the theory. 38

Table 2: Benefits of the theory, mapping to aspects and demonstrations. 38

Abstract

This dissertation presents a theory describing the components of a cognitive architecture supporting intelligent behavior in spatial tasks. In this theory, an abstract symbolic representation serves as the basis for decisions. As a means to support abstract decision-making, imagery processes are also present. Here, a concrete (highly detailed) representation of the state of the problem is maintained in parallel with the abstract representation. Perceptual and action systems are decomposed into parts that operate between the environment and the concrete representation, and parts that operate between the concrete and abstract representations. Control processes can issue actions as a continuous function of information in the concrete representation, and actions can be simulated (imagined) in terms of it. The agent can then derive useful abstract information by applying perceptual processes to the resulting concrete state.

This theory addresses two challenges in architecture design that arise due to the diversity and complexity of spatial tasks that an intelligent agent must address. The perceptual abstraction problem results from the difficulty of creating a single perception system able to induce appropriate abstract representations in each of the many tasks an agent might encounter, and the irreducibility problem arises because some tasks are resistant to being abstracted at all. Imagery works to mitigate the perceptual abstraction problem by allowing a given perception system to work in more tasks, as perception can be dynamically combined with imagery. Continuous control, and the simulation thereof via imagery, works to mitigate the irreducibility problem. The use of imagery to address these challenges differs from other approaches in AI, where imagery is considered as an alternative to abstract representation, rather than as a means to it.

A detailed implementation of the theory is described, which is an extension of the Soar cognitive architecture. Agents instantiated in this architecture are demonstrated,

including agents that use reinforcement learning and imagery to play arcade games, and an agent that performs sampling-based motion planning for a car-like vehicle. The performance of these agents is discussed in the context of the underlying architectural theory. Connections between this work and psychological theories of mental imagery are also discussed.

Chapter I - Introduction

People are confronted with a range of situations in their everyday lives that are characterized by a need for precise interaction with the spatial aspects of their surroundings. As a few extreme examples, consider catching a ball, solving a jigsaw puzzle, or parallel parking. To catch a ball, a person must position their hand in a place where the ball will arrive; whether or not a given position meets this criterion depends upon the exact velocity of the ball and the influence of gravity. To solve a puzzle, a person must find which pieces fit together, which is a property that depends on the precise details of the shapes of both pieces. And to parallel park a car, the complex relationship between the controls of the car and its position on the street determines whether or not a given action sequence will result in successful parking.

Of course, the fact that a person can act as if all of this information has been considered does not imply that it is explicitly represented within their mind and reasoned over. However, to build machines capable of human-level intelligence, or to create detailed models of human cognition, hypotheses about what is represented and how it is processed to generate this sort of behavior are needed.

In this thesis, work developing such a theory is presented. The problems I focus on are spatial control problems. In these problems, based on the spatial state of the world, an agent must make decisions about actions that will (possibly) change that state. This category includes tasks like those mentioned above, along with simpler tasks like the stacking blocks on a table. It does not include tasks where the perceptions provided to the agent are not spatial, like language understanding, or tasks where the result of reasoning is not the selection of an action that will have spatial effects, like solving a physics homework problem. However, those tasks *do* involve spatial information, and humans address all of these tasks (seemingly using the same basic machinery), so it is a

long-term goal of the theory to eventually address them. Spatial control problems are some of the most basic problems any agent must solve, though, so they will be the focus here.

In spatial control tasks, as in most tasks, an agent can benefit from using an abstract internal representation of the structure of the task, where unnecessary detail is removed, allowing for more efficient processing. However, as will be explained in detail in the next chapter, there are two problems inherent in designing an architecture capable of abstract representation in spatial control tasks. First, the diversity of tasks an intelligent agent must address is large, and it is difficult to create a single perception system to create appropriate abstract representations in all such tasks. This difficulty is the *perceptual abstraction* problem. Second, some tasks are resistant to being abstracted at all, as is the case when the appropriate action outputs vary continuously as a precise function of the details of the environment: this is the *irreducibility* problem. The use of continuous controllers can partially remediate the irreducibility problem, but difficulties are encountered integrating these into a general-purpose cognitive architecture when the diversity and complexity of different spatial tasks is taken into account.

In this thesis, I present a theory of basic architectural mechanisms that can work to mitigate the perceptual abstraction and irreducibility problems. By mitigating these problems, the central claim of this thesis is that this theory represents progress towards the goal of a cognitive architecture capable of general intelligence in spatial tasks. The crucial aspects of the theory include the use of both abstract and concrete (highly detailed) representations of the state of world, continuous action controllers which access the concrete representation, and simulative imagery capability, where internal simulations based on concrete representation are used to derive abstract information about the consequences of potential actions.

As will be explained, the theory leads to a set of functional benefits, which act both to improve performance in individual tasks and to increase the number of tasks that can be

addressed by the architecture (its generality). The theory has been implemented, by augmenting the Soar cognitive architecture (Laird, 2008; Newell, 1990) with memories and processes for handling spatial information. Agents instantiated in the implemented architecture presented here provide demonstrations of both the operation of the architecture itself and the functional benefits of the underlying theory.

An inspiration for the theory has been psychological research in mental imagery. This research has provided strong evidence that people maintain and manipulate visual and spatial information at a level close to that of perception, reusing the same systems that process perceptual data to process internally generated (imaginary) data (Kosslyn et al., 2006). This work in this thesis builds on existing work on computational imagery systems, particularly that of Lathrop (2008), who created a pilot implementation of a mental imagery extension for Soar, but also drawing on other theories and systems (e.g., Barsalou, 1999; Glasgow & Papadias, 1992; Grush, 2004; Huffman & Laird, 1992; Kosslyn et al., 2006; Kurup & Chandrasekaran, 2006).

However, as outlined above, here, I start with the assumption that abstract representation is used, and examine issues that arise in creating an architecture to support abstract representation in arbitrary problems. Concrete, perceptual-level representation and imagery are incorporated in the theory as a means to this end. In contrast, in other theories and systems, these aspects are instead presented as a means to model psychological phenomena, or to allow the agent to exploit differences in processing efficiency that different representations allow.

To elaborate on this, prior functionality-based examinations of imagery have assumed that since, in principle, abstract propositional representations and concrete perceptual representations can encode the same information, the primary functional role for imagery is to allow more efficient inference. However, the analysis here reveals that, in a general-purpose architecture, these representations will likely not be informationally equivalent. Particularly, the abstract representation alone cannot capture all relevant

details of the problem, while these details can be represented at the concrete level. A functional role for imagery is then to compensate for this informational inequivalence.

This concept leads to a difference in the broad way perceptual-level representation and imagery are understood in the context of a cognitive system. Rather than viewing these aspects primarily as a more efficient means for addressing particular tasks, like solving geometry problems, or as a means to model human imagination, here, they are viewed as essential parts of the basic process of capturing the right details of the state of the outside world in order to choose an action.

This view is reflected in the types of tasks addressed. Spatial control tasks are studied, rather than tasks where the goal itself is to imagine something. The use of imagery in a particular task emerges from the need to construct appropriate abstract properties for that task given the architectural means available, not due to the task being "about" imagery. The same basic architectural mechanisms that allow an agent to perform mental rotation or solve geometry problems might also allow this, but the tasks examined here are more basic, involving the immediate interaction between the agent and its environment.

In order to make traction in these tasks, in this work, it is assumed that a concrete representation encoding spatial properties is available, and that this representation, combined with the agent's background knowledge, contains all of the relevant information necessary for the agent to act intelligently (where imagery processing may be necessary to make some of this information explicit). General-purpose perception in AI and robotics is an unsolved problem, so in this work the tasks studied will use either simulated environments or limited environments where perception is possible.

Nevertheless, as will be demonstrated, interesting tasks can still be addressed, and progress can be made towards the overall goal of a general-purpose cognitive architecture for spatial tasks.

The plan for the rest of this document is as follows:

Chapter II outlines the basic assumptions behind this research, and then motivates the perceptual abstraction and irreducibility problems through a few detailed examples.

In Chapter III, the basic architectural theory is introduced, and mapped to particular functional benefits. The approach for evaluating the theory is also explained in this chapter.

In Chapter IV, the Soar/SVS architecture which instantiates the theory is presented. As a complete, functional system, this architecture includes many important details that are not part of the general theory, including mechanisms for several different types of imagery operations. These details are described here.

Chapter V presents a set of reinforcement learning agents instantiated in Soar/SVS. A simple technique is presented, where imagery is used as a means to infer abstract state information in a reinforcement learning agent, and a new algorithm, ReLAI, is introduced that more tightly integrates imagery and reinforcement learning. Some basic theoretical analysis of ReLAI is presented, and related to the benefits of the theory. An interface connecting Soar/SVS to an emulator for the Atari 2600 game system is briefly discussed, and agents that address tasks based in three different arcade games are presented. These agents are analyzed with respect to the ReLAI algorithm and with respect to the broader architectural theory.

In Chapter VI, an agent in Soar/SVS that performs motion planning for a car-like vehicle is presented. This agent is an instantiation of an existing algorithm for motion planning within the architecture. The algorithm, RRT (LaValle & Kuffner Jr., 2001) was developed by others not working in the context of a cognitive architecture, but maps easily onto the system here, and demonstrates the benefits of the theory.

Chapter VII presents related work, including discussions of AI research in cognitive architecture, robotics, and qualitative reasoning, along with psychological research in cognitive modeling and theories of grounded cognition.

Chapter VIII reviews the claims made and the demonstrations thereof in implemented agents. A brief discussion is presented, covering how insights uncovered here might apply to cognitive models and the imagery debate in psychology. Some contributions of the thesis are enumerated, future work is discussed, and the main body of the dissertation is concluded.

At the end of the document, an appendix is presented where issues involving the representation of quantities in Soar's working memory are discussed, as many concerns in that area overlap with the work on SVS presented in the main part of the thesis.

Chapter II - Motivation

In this chapter, issues behind representing and solving spatial problems in a task-independent architecture are introduced through a few simple examples, and some basic problems in cognitive architecture design are laid out.

2.1 Using Symbolic Representation for Decision-Making

At a high level, any agent can be viewed as a system that takes in sensations and produces actions. The central goal of this work is to investigate cognitive architectural structures to support intelligence in spatial tasks; that is, to investigate common computational representations and mechanisms to allow an agent to efficiently process its sensations and produce action choices that an external evaluator might judge as intelligent.

The space of all theories is huge, and not all possibilities can be addressed here. Instead, this analysis takes discrete decision-making as a starting point: we assume that the agent's reasoning process is a series of steps where potential action choices are weighed, and one action is chosen. In addition, we assume that this decision-making is contingent upon symbolic information. To keep the discussion as general as possible, the definition of "symbolic information" will remain purposefully vague. Here, the only important property of this information is that, from the perspective of the decision procedure, symbols are discrete entities that have no intrinsic similarity to one another. Essentially, this means that properties influencing the decision (for example, learned action values) cannot be a continuous function of components of the agent's internal state.

As will be explained in detail shortly, there are two basic implications of these assumptions: an agent needs to derive symbolic information that distinguishes between situations where one decision should be made versus another (so the correct decisions

can be made), and this symbolic information should distinguish between as few situations as possible (so minimal knowledge is required to make decisions).

These assumptions are fulfilled by decision making in Soar and other symbolic cognitive architectures, along with table-based reinforcement learning systems and symbolic planning systems. They do not cover reinforcement learning with continuous function approximation, nor systems where the agent's entire behavior is described by a continuous function from input to output (e.g., a feedback rule or neural network), and there is no notion of discrete decisions between actions.

It should be noted that these assumptions do not mean that actions must be a function of symbolic perceptions alone. Previous perceptions and arbitrary background knowledge can influence decision making. In addition, non-symbolic processes can operate over symbolic information and effect decision making. For example, reinforcement learning adjusting control biases (Sutton & Barto, 1998), memory activation influencing knowledge retrieval (Anderson et al., 2004), or Bayesian reasoning to infer properties from evidence (Tenenbaum et al., 2006) can all fit in this framework.

In addition, the assumption that discrete decision-making is present is not intended to mean that every aspect of the agent's external behavior—every detail of its motor vector—must be the direct result of a decision. As will be discussed, hierarchical control is necessary in many spatial problems, and symbolic decision-making at the upper level(s) of the hierarchy is sufficient to meet these assumptions.

2.2 Motivating Tasks

To better understand what is necessary to represent and solve spatial tasks, three example tasks are introduced in this section.

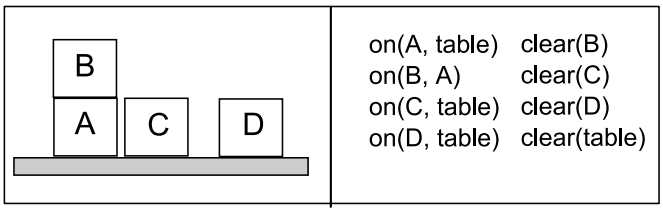

Figure 1: A simple blocks world task.

2.2.1 The Blocks World

In the blocks world (Figure 1), an agent is presented with some blocks on a table, and has a simple task of stacking them in some specific configuration, such as block **A** on block **B** on block **C**. At any time, the agent can move any block that has nothing on top of it. Blocks can be moved to the top of another block or to the table.

A straightforward way of addressing this task in a symbolic agent is to use a planning language such as STRIPS. The state is described in terms of abstract predicates[1], as shown on the right of the figure, and rules encode possible actions available to the agent. For example, a rule might encode that if a block **X** is on another block **Y** (**`on(X,Y)`** is true), and **X** moved to the table, block **Y** is now clear and block **X** is now on the table (**`on(X,Y)`** is no longer true, **`on(X,table)`** is true, and **`clear(Y)`** is true). In the simplest case, the initial state and the goal of the problem are expressed in similar terms, and the problem space can be searched through using a standard algorithm (e.g., iterative deepening), finding a sequence of actions that lead to the goal.

An alternative approach is to use a reinforcement learning algorithm to gradually learn a policy through interaction with the environment (Sutton & Barto, 1998). In this approach, the state could again be represented in terms of abstract predicates[2], but the goal is instead mapped onto a reward signal. With enough trials, the agent can learn a policy to maximize its reward, effectively solving the problem.

---

[1] Throughout this work, predicate representations will be used as notational shorthand for generic symbolic representations. The use of predicates is not meant to imply anything about the symbolic processing that could be used, but merely to illustrate what data is encoded in terms of symbols.

[2] In line with the symbolic decision-making assumption, when discussing reinforcement learning here, I am referring to table-based RL, and not RL with function approximation. Connections between the issues discussed here and RL function approximation will be discussed in Section 7.4.

These agents share the general characteristic that the end result (after planning has occurred, or a policy has been learned) is a mapping from states to actions. In the blocks world, for both of these agents, the **on** and **`clear`** predicates capture the right aspects of the task such that an optimal plan or policy can easily be found by an appropriate planner or learning algorithm.

One of these agents might exist in a world where it is repeatedly presented with blocks world situations, and must solve each one. In that case, the agent is solving multiple *instances* of the same task. In realistic environments, any given instance of a task might vary in its details. For example, in different instances, the blocks might be in slightly different positions, they might be different colors, or be slightly larger or smaller.

However, the representation of the task used here is good enough that it covers all of these variations of the task. This is the benefit of using an abstract description of the world. If an abstraction correctly summarizes the important properties of the task, as it does in this case, lower-level details can be ignored so that many underlying problem instances are mapped to a single internal task representation. Any instance of the task where the initial state encodes that all blocks are on the table can be solved with the same action sequence, regardless of the exact location of the blocks on the table, their size, color, etc.

In opposition to the abstract agents described above, consider an agent using a more detailed representation, such as continuous coordinates describing the shapes of blocks, without a higher-level interpretation (we will call this detailed representation *concrete*). As these coordinates would be treated symbolically, any variation in the blocks (however minor) would cause the agent to perceive a completely different state. An agent using concrete information for a symbolic representation would have very little generalization ability: it would be extremely unlikely that two blocks in different problem instances would ever appear in the same precise position, so the agent would never be able to transfer knowledge between instances of the task, no matter how similar they may appear to an observer.

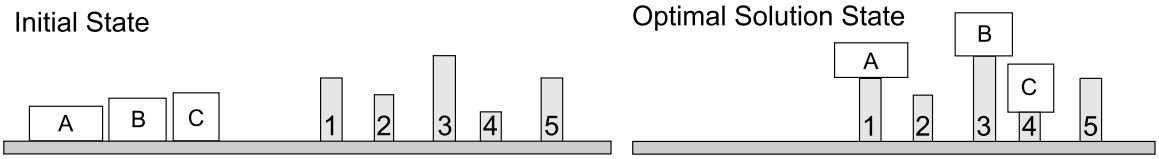

Figure 2: A pedestal blocks world problem and its optimal solution state.

2.2.2 The Pedestal Blocks World

Now, consider a slightly modified version of the environment (Figure 2). Here, the agent is presented with a table and three blocks. There are some pedestals fixed to the table upon which the blocks can be placed. The goal is to place the blocks on the pedestals in the correct order (**A** to the left, then **B**, then **C** to the right). The agent moves each block to a pedestal, first **A** then **B** then **C**.

Rather than having a single goal, this agent receives a numerical reward proportional to the quality of its solution. A reward of 100 is received for placing the blocks in the correct order. It is better to place the blocks as far to the left as possible: 10 points are deducted from the reward for each empty platform to the left of **C**. However, the blocks can only be placed centered on the pedestals (otherwise they fall off), and the pedestals may be positioned such that a certain block cannot be placed on a certain pedestal, as a neighboring pedestal is in the way, or that two blocks cannot fit on adjacent pedestals. If the agent places a block where it cannot fit, it receives a reward of -100 and the task ends. If the agent places the blocks on the pedestals without a collision, but the ordering is wrong, a -10 reward is received.

This task is not as straightforward for a symbolic agent to address, as it is not as clear how to represent the state of the task in terms of abstract symbols. If symbols simply describe the same basic aspects of the state as was necessary in the unmodified blocks world (**on** and **`clear`** predicates), these symbols are insufficient to distinguish between cases where the best action is, for example, moving **A** to **`pedestal1`** from cases where the best action is to instead move **A** to **`pedestal2`**. This is because the crucial aspect of the problem that affects which choice is better is not captured by the given symbols: whether or not the given action would cause a collision.

Instead, more complex predicates are needed. For example, a predicate encoding exactly the relevant property would suffice: for example **`collision_if_moved(A,pedestal1)`** might be true if moving block **A** to **`pedestal1`** would cause a collision. However, note that this predicate is not a simple property describing the state of the world, like the fact that a block is on the table, but rather encodes a much more complicated, task-specific relationship. For example, Figure 3 shows a situation where, to infer **`collision_if_moved(C,pedestal2)`**, the exact sizes and positions of three blocks and three pedestals need to be accounted for, and one of those blocks (**C**) is located spatially far from the other objects. Overall, the symbolic information necessary in this task is not a simple, local property of the world that one might expect a generic perception system to calculate.

2.2.3 Motion Planning for a Nonholonomic Car

One final task is needed to introduce a relevant aspect of intelligent spatial behavior that is not apparent in blocks world examples: precise control. As a representative task of this sort, the problem of motion planning for a nonholonomic car will be considered.

Even in the blocks world, if a real robot is used, precise control is necessary. In such a system, the final output of the agent is a set of motor voltages. Since real blocks can vary in size and shape, the actual voltages output might need to be sensitive to those variations. If the motor voltages are continuous, as are the positions of the blocks, the problem likely cannot be solved by symbolically mapping abstract symbolic perceptions to actions.

However, in simple domains, this aspect can be ignored. A complete system would

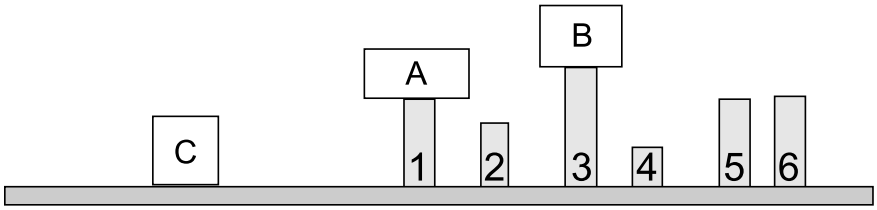

Figure 3: A situation in pedestal blocks world where nonlocal interactions are important if the agent is considering moving block C to pedestal 2.

include low-level controllers in the architecture that continuously transform the perceived state to output voltages. For example, for a motor that causes a translation of a gripper arm, the output voltage could be set to $\alpha\Delta_x$, where Δ_x is the distance from the arm to the block and α is a negative constant factor, in order to move the gripper to the block. This is a partially nonsymbolic approach to the problem, but fits within a system using symbolic representations for decision making: controllers can be encapsulated in modules isolated from the rest of cognition, and the actions of the agent can simply be viewed as selecting between controllers (e.g., Laird et al., 1991).

For this encapsulated controller approach to work, though, to make intelligent decisions, the controllers typically must have symbolic characterizations: the behavior of the controllers must result in consistent transitions between symbolic states. This essentially means that the controllers must have performance guarantees.

For example, a robotic blocks world agent might have a controller to move a gripper arm to a block and grasp it. If every time **`clear(A)`** is true, invoking the controller results simply results in the agent picking up block **A**, the agent can act intelligently. If in some circumstances reaching for a block might also knock down towers of nearby blocks, the agent needs to have symbolic information that allows it to distinguish those circumstances if it is to make intelligent decisions.

Motion planning, as it is considered here, is the problem of determining a sequence of control outputs that causes a robot to move through space to reach a goal position.[3] In Figure 4, the task is to drive the car object to the goal region while avoiding the grey obstacles. A line outlining the optimal path to follow is shown. Some approaches to motion planning use encapsulated controllers, where low-level controllers are designed such that the problem can be reduced to a search through the symbolic states those controllers reliably traverse (e.g., Beeson et al., 2010). This works well for particular

[3] The problem is posed slightly more generally in the motion planning literature, but the distinction is unimportant here.

classes of robots, such as polygonal robots that can move in any direction, and for more complicated robots when tight maneuvering is less important.

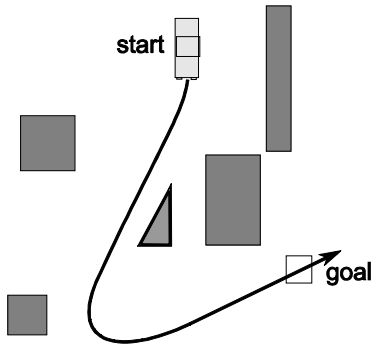

Figure 4: A nonholonomic car motion planning problem.

However, in other situations, the encapsulated controller approach does not work as well. One reason for this difficulty is that certain kinds of constraints on motion are infeasible to capture in abstract representations, and creating an abstract representation is necessary for the encapsulated controller approach to work well. Nonholonomic constraints result from systems where the number of controllable dimensions is less than the total number of degrees of freedom. For instance, a car is nonholonomic, since its position can be described by three parameters (its position on the surface of the earth and the direction it is facing), but it is only controllable in two dimensions (driving forward and reverse, and steering left and right). Where it is relatively straightforward to abstractly characterize the relevant parts of the reachable space of a robot that can turn in place, this is not as simple with a car-like robot when precise details matter. This difficulty is familiar to most drivers: it is relatively easy to determine a sequence of roads to travel on to reach a destination (where precise details beyond the specific roads traveled on do not matter), but it is much more difficult to parallel park (where the details do matter). The figure shows an example problem where the nonholonomic constraints matter. It is difficult to come up with an abstraction of the problem where the path shown could have been composed by searching through an abstract state space.

This situation, where precise control cannot be reduced to transitions between symbolic states, presents a challenge to symbolic agents, which will be addressed shortly.

2.3 Meta-Problems in Architecture Design

These domains provide insight into some fundamental issues in cognitive architecture design. These issues will be presented as meta-problems: problems that the design of the architecture must support solving.

First, in order behave intelligently in any task, the agent needs to use its perception system to infer information about the outside world, which leads to the first meta-problem:

Veridical Perception Problem: An agent must have means to use its perceptual input to determine sufficient information about the true state of the world in order to intelligently select actions.

The notion of "intelligently" selecting actions will be discussed shortly.

This problem is posed mainly to clarify what this research is *not* about. Much research in AI and related fields is working towards addressing veridical perception, including research in robotic perception and in computer vision. However, the focus here is instead on problems that arise even when the complete state of the world is known to the agent, such as it is in virtual environments. The previous section described difficulties related to perception in both the pedestal blocks world and nonholonomic motion planning tasks, however, these difficulties had nothing to do with inferring the true state of the world: they had to do with representing that information in a form such that actions can be chosen.

Another meta-problem is then present, related to the need for an agent to construct appropriate abstract symbols to choose actions. Any agent that performs symbolic decision-making needs to derive symbols from its perceptual input that it can use to distinguish between situations where one action should be chosen versus another. If two situations cannot be distinguished in terms of the available symbolic information, the agent will make the same action choice in both, regardless of what non-symbolic processes are operating in terms of the symbolic representation. For example, an agent

in the pedestal blocks world using only **on** predicates cannot distinguish between states where moving block **A** to **pedestal1** will and will not cause a collision.

In addition, these symbols should distinguish as *few* states as possible—they should be abstract. If, instead of using **on** predicates, the pedestal blocks world agent encoded every detail of the problem in its symbolic representation (a concrete symbolic representation), planning or learning would be extremely difficult, especially if multiple instances of the problem were addressed.

Any agent architecture following the symbolic assumption must then solve a problem of perceptual abstraction:

Perceptual Abstraction Problem: An agent must have a means to create abstract symbolic structures from perceptual input that can serve as the basis for intelligent action choices.

The discussion of control in the previous section motivates another meta-problem. If all behavior is simply viewed as mapping primitive perceptions to symbolic information, and selecting primitive actions based on that symbolic information, there may be no possible symbolic representation of the problem that makes all of the necessary distinctions between situations and yet is abstract enough that efficient planning or general learning is possible.

Irreducibility problem: An agent must have the means to intelligently act in tasks where abstract, purely-symbolic representation is not possible.[4]

In many cases, the irreducibility problem can be handled by including encapsulated controllers in the architecture, as can be done in the blocks world. In that case, at the

[4] The word "irreducibility" here makes the most sense when the task is viewed as an MDP (as will be discussed in Chapter V). The size of an MDP may be reduced by identifying equivalent states and/or actions and combining them (Givan et al., 2003). However, at some point the MDP will reach a minimal size. If the minimal MDP is still very large, we call the problem irreducible. In general, though the term can be used in the context of any symbolic representation scheme, not just MDPs.

symbolic level, the actions are to choose among controllers rather than to issue raw motor commands. However, in other tasks, such as motion planning for a nonholonomic vehicle, this simple reduction may not be possible, as there is no apparent way to effectively divide the problem between low-level controllers and symbolic reasoning to choose between controllers.

The word "intelligence" is prominent in all three of these problems. The broad goal of this work is to work towards an implemented cognitive architecture capable of human-level intelligence in spatial problems. Towards this goal, intelligence should be judged based on matching human behavior. While it is possible that the definitions of these problems can vary based on differences in what it means to "match" human behavior, and differences in the details of the agent, for the level of analysis presented here these differences should be minor. Moreover, in the tasks examined here, matching human behavior and achieving optimal performance will not be differentiated—progress toward optimal performance will be used as a substitute for progress toward human-level performance. The goal of matching human behavior here serves only as a very rough guide to what tasks should be addressed and what level of performance should be achieved, rather than as a direct objective to evaluate.

Another important aspect that affects the difficulty of solving these meta-problems is the number of tasks the agent is to address. Solving the problems and creating an architecture capable of supporting human-level performance in a single task is much simpler than doing the same for a general-purpose architecture. Accordingly, the *general* veridical perception, perceptual abstraction, and irreducibility problems are defined to be the versions of these problems faced that must be addressed by an architecture capable of supporting intelligent behavior in the same breadth of tasks as humans.

2.4 Imagery for Spatial Tasks

In this thesis, cognitive architecture structures are introduced which are proposed to work towards solving the general perceptual abstraction and general irreducibility

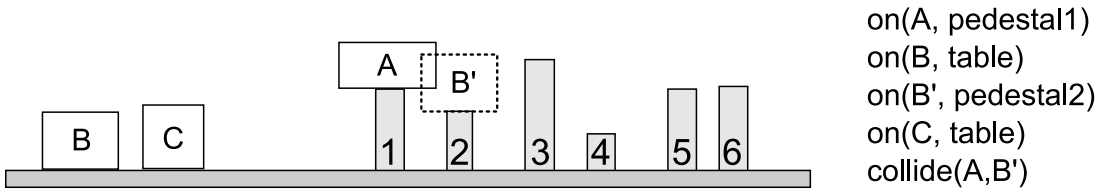

Figure 5: Imagery in pedestal blocks world.
The agent has imagined block b on pedestal2 (creating an imagined copy, B') and inferred the abstract predicates at right.

problems. These structures support what is called simulative imagery. Here, simulative imagery is explained at a high level in the context of two of the tasks introduced in Section 2.2.

In the pedestal blocks world, an issue with symbolic representation was that some important information—the circumstances under which a given action will cause a collision or not—is difficult to capture in terms of symbols. A predicate to capture this information was proposed (**`collision_if_moved`**), however, calculating this predicate involves many factors, and it is not obvious how a task-independent agent could infer it. That is, it is one of the difficult cases that make general perceptual abstraction a problem.

In this thesis, the proposed solution to the difficulty of perceptual abstraction in cases like these is to use imagery. An imagery agent has both an internal abstract problem representation, along with a more precise internal concrete representation: a representation that makes as many distinctions as possible between states of the world. That is, it has internal representations akin to both pictures and predicates. System design is covered later, but for this discussion, it is sufficient to say that the agent can simulate its actions in terms of the concrete representation, and derive the resulting abstract state. In the pedestal blocks world, the agent can imagine what would happen if it were to move a given block on to a given pedestal, and detect whether a collision would result (Figure 5). Essentially, imagery here allows a complex, task-specific predicate to be inferred by using a combination of simple, task-independent mechanisms.

In nonholonomic motion planning, the issue discussed above is that the selection of complex control sequences cannot easily be reduced to a search through abstract symbolic states. That is, the problem is irreducible, and this irreducibility cannot be handled by encapsulated controllers. In response to this difficulty, a common approach used is *sampling-based* motion planning (Lindemann & LaValle, 2003). These techniques determine the reachable locations for a robot by simulating motion from its current position. This simulation process can be considered a form of imagery (Wintermute, 2009a). Sampling-based motion planning is often used in conjunction with low-level controllers. For a car planning problem, a controller can be created to steer the car toward a point in space, and the algorithm samples possible inputs to this controller (intermediate goal points, or waypoints) through simulation to find a sequence that results in a short, collision-free path reaching the goal. Leaving aside (for now) the issue of how candidate waypoints are generated, the relevant aspect of this technique is that imagery operations simulating the behavior of the agent's low-level controllers are an essential part of this motion planning technique. In many cases, the actual controllers used for external action can be run on simulated data to allow this (e.g., Leonard et al., 2008).

In using imagery, the problem is then divided between a high-level search over possible sequences of waypoints, and low-level simulations over concrete states that determine which further waypoints are reachable from a known state. This differs from the encapsulated controller approach: while the technique still has aspects of a search through abstract states, the problem is not *reduced to* such a search. Abstract states encoding information like "reached waypoint 12" or "collided with an obstacle" are used, but the agent has no way of knowing what future abstract state transitions will happen without using simulative imagery. Put another way, the agent can use low-level controllers whose behavior cannot be reliably characterized with simple abstract state transitions. Through the use of simulative imagery, though, the irreducibility problem is mitigated in these agents. A complete motion planning agent will be discussed in Chapter VI.

Chapter III - A Theoretical Architecture for Spatial Problems

The examples in the previous chapter informally present some aspects of an architecture for spatial tasks and why they are useful, but there are still many unanswered questions. In this chapter, a theory for an architecture incorporating these aspects is described, and specific functional benefits afforded by the theory are described. In Chapter IV, details of a computational instantiation of the theory are described.

3.1 Theory Description

Many types of AI systems fit the basic pattern that perceptions are mapped to an abstract problem state, and abstract decision-making occurs in terms of that problem state. This is shown in Figure 6(a). In the figure, the decision system could be a symbolic planner or a reinforcement learning system, or something less constrained such as Soar's symbolic processing.

Details of perception are often ignored when discussing these systems and only the

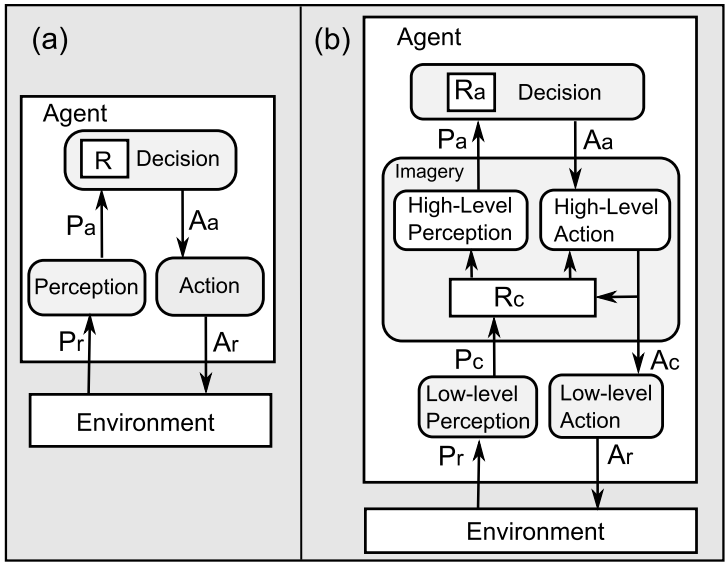

Figure 6: A basic non-imagery architecture (left) and an imagery architecture

internal problem representation is addressed, but a perception algorithm is still at least implicitly part of the system. For example, in the blocks world, the internal representation could consists of predicates like **on(A,B)** and **clear(C)**, and in an embodied agent, some sort of vision system would build those predicates during perception.

Figure 6(a) labels the different parts of this generic architecture. Call the direct output of the agent's sensors $\boldsymbol{P}_r$, for raw perception. This signal is transformed by the perception system to create an abstract perception signal, called $\boldsymbol{P}_a$. The system maintains an internal abstract representation of the problem state, $\boldsymbol{R}$, calculated as a function $\boldsymbol{P}_a$, possibly taking into account past observations and background knowledge. Agents of this sort also typically use a high-level representation of actions: it is rare that actions are considered in terms like "set motor voltage to .236", even though that may be the final output of an embodied agent. So, even in a simple system, there are typically distinct abstract and raw action signals, $\boldsymbol{A}_a$ and $\boldsymbol{A}_r$, and a motor system that creates $\boldsymbol{A}_r$ from $\boldsymbol{A}_a$.

An architecture with imagery is shown in Figure 6(b). A concrete representation, $\boldsymbol{R}_c$, is present, in addition to $\boldsymbol{R}_a$ (in the abstract decision system). An additional level of perceptual and action processing has also been added. The output of low-level perception is now provided to $\boldsymbol{R}_c$, so it is called $\boldsymbol{P}_c$, for concrete perception. $\boldsymbol{R}_c$ is chiefly derived from this signal. However, $\boldsymbol{R}_c$ is not strictly a reflection of $\boldsymbol{P}_c$, but can also be locally manipulated. In particular, it can be manipulated based on the high-level action signal $\boldsymbol{A}_a$ from the abstract decision system. High-level perception processes transform $\boldsymbol{R}_c$ into $\boldsymbol{P}_a$, which is the perception signal provided to the abstract decision system. Note that this happens independently of whether the contents of $\boldsymbol{R}_c$ are real or imagined: the form of $\boldsymbol{P}_a$ is the same, just possibly annotated as real or imagined.

Imagery actions share common mechanisms with external actions. Agents can thus simulate the results of external actions in the imagery system. Moreover, the system can now use actions that cause changes to the imagery system, but do not have a

corresponding external action. These imagery actions can be used for many different things, for example, memories could be retrieved, or geometric reasoning could be performed. For this discussion, though, we will focus on simulative imagery: using imagery actions to predict the value of $\boldsymbol{P}_{\mathbf{a}}$ a given action would cause if it were to be executed in the environment. Through simulative imagery, the abstract decision system can get information about the state of the world not just via $\boldsymbol{P}_{\mathrm{a}}$ directly, but via predictions about *future* values of $\boldsymbol{P}_{\mathrm{a}}$. Both these predictions and the execution of the external actions themselves can be based upon information not present in the current value of $\boldsymbol{P}_{\mathrm{a}}$ itself, but present in $\boldsymbol{P}_{\mathrm{c}}$.

The properties of an architecture following this theory can be divided into nine important aspects, as outlined below. Each of these aspects (excluding the first) can depend on others being present (as indicated), otherwise they are independent of one another. This theory is being described with a sort of open world assumption: other aspects not described here may also be present, unless they are specifically ruled out.

A1. Bimodality

- Two representations of information derived from perception are present, $\boldsymbol{R}_{\mathrm{a}}$ and $\boldsymbol{R}_{\mathrm{c}}$.
- Representation $\boldsymbol{R}_{\mathrm{c}}$ contains more perceptual information than $\boldsymbol{R}_{\mathrm{a}}$—it makes more distinctions between states of the world. If $\boldsymbol{R}_{\mathrm{c}}$ encodes spatial locations of objects in the world, it is a *concrete spatial* representation.[5]
- Processes can encode information in $\boldsymbol{R}_{\mathrm{a}}$ based on $\boldsymbol{R}_{\mathrm{c}}$ (through *high-level perception* processes).

A2. Concrete routines

Requires A1.

[5] While the focus here is on spatial information, this theory could potentially also apply to non-spatial (e.g., auditory) modalities, although that possibility will not be covered here. In addition, other information can be present in the concrete representation that may not be considered "spatial". The implemented system discussed in Chapter IV includes object shape, velocity, and identity information in the concrete spatial representation, and other systems could include information like the mass or acceleration of objects.

- Processes can cause changes to representation $\boldsymbol{R}_c$ based on its existing contents (they can locally manipulate it).

A3. Imagery

Requires A1, A2.

- Concrete routines can be invoked by processing in $\boldsymbol{R}_a$, and result in persistent changes to $\boldsymbol{R}_c$. These are imagery processes.
- Via high-level perception, results of imagery are reflected in $\boldsymbol{R}_a$.

A4. Simulative imagery

Requires A1, A2, A3.

- Some imagery operations simulate future states of the world in terms of $\boldsymbol{R}_c$: they manipulate $\boldsymbol{R}_c$ such that its resulting state is similar to a situation that might be perceived in the future.

A5. Concrete controllers

Requires A1.

- External actions can be contingent on information in $\boldsymbol{R}_c$ but not in $\boldsymbol{R}_a$. Modules that generate these actions are called concrete controllers.

A6. Simulative imagery of concrete control

Requires A1, A2, A3, A4, A5.

- Some simulative imagery operations simulate the effects of concrete controllers.

A7. Architectural representation conversion

Requires A1.

- High-level perception and imagery are supported by specialized architectural mechanisms.

A8. Perception/action reuse

Requires A1, A7.

- Some types of perceptual information[6] arrive in $\boldsymbol{R}_a$ only via $\boldsymbol{R}_c$.

[6] For example, if the concrete representation encodes spatial information, spatial information may only arrive in the decision system via that representation. This is not *all* perceptual information, though, since the agent might have different modalities of perception (e.g., visual and auditory) that might vary in their imagery capability.

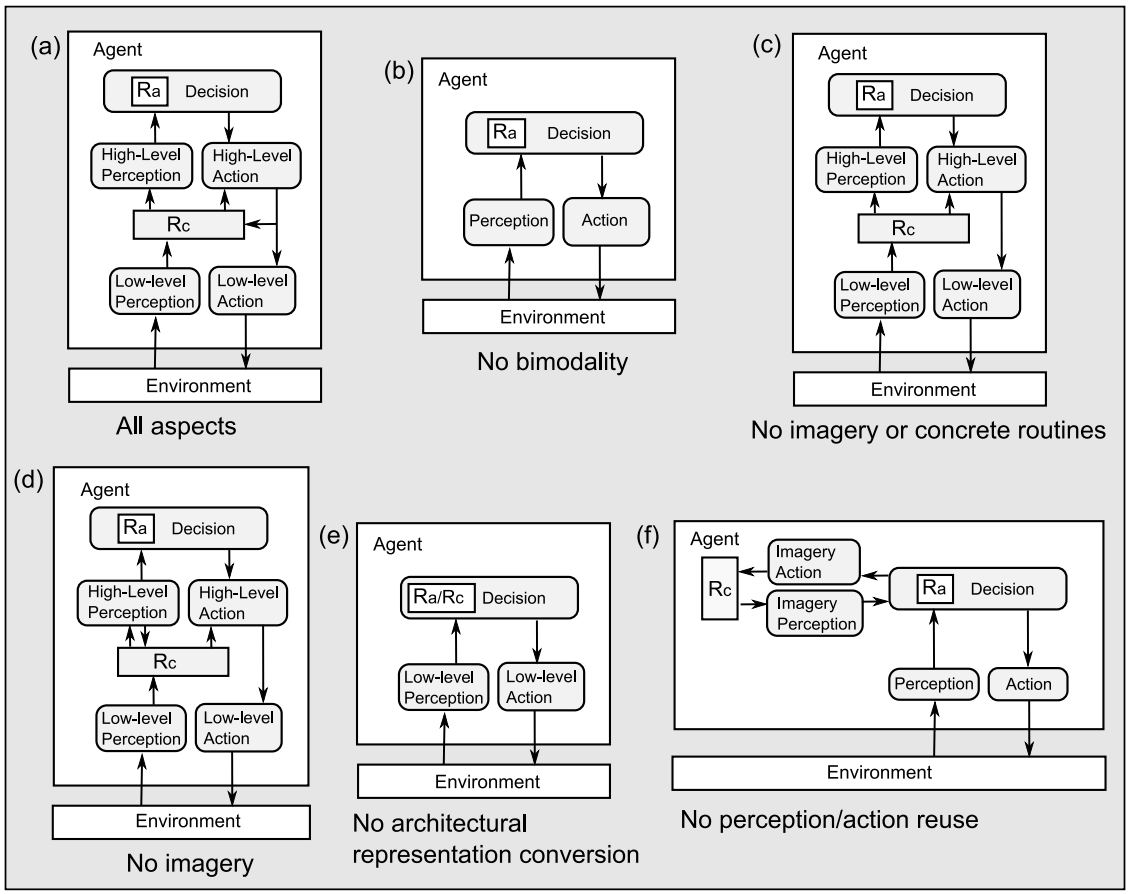

Figure 7: Alternative architecture designs lacking aspects of the imagery theory.

- Common high-level perception mechanisms operate over structures created by imagery and low-level perception in $\boldsymbol{R}_C$.
- Some imagery processes share mechanisms with those used to generate external actions.

As will be discussed in the next chapter, each of these aspects maps in a different ways to functional benefits. To clarify what how each aspect affects the overall architecture, alternative systems can be described that lack that aspect. Some of these systems are shown in Figure 7 The implications of leaving out these aspects, however, are not discussed here, as that will be covered in the next section.

Figure 7(a) shows a system including all of the aspects. Figure 7(b) demonstrates a system lacking in bimodality. This system only represents information at one level of abstraction.

Figure 7(c) shows an example of a bimodal system without imagery or concrete routines. Here, the high-level action system has no connection to $\boldsymbol{R}_{\mathrm{C}}$. This is a simple hierarchical control system, where the only role of the concrete representation is to provide more state information for controllers in the action system.

A bimodal system can have imagery capability without simulative imagery. Many previous computational imagery systems address tasks that are more about high-level reasoning than detailed interaction with a spatial environment, such as solving geometry problems (e.g., Gelernter, 1963) or geographic reasoning (e.g., Barkowsky, 2002) and hence lack this aspect.

Figure 7(d) shows an example of a bimodal system with concrete routines, but not imagery. In such a system, high-level perception uses concrete routines as a means to derive abstract properties. Hence, there is an arrow from that system to $\boldsymbol{R}_{\mathrm{C}}$ rather than from the action system. Such a system can include concrete routines that simulate actions. Concrete routines are a generalization[7] of Ullman's concept of visual routines (Ullman, 1984). These are local processes within a concrete visual representation that are used as means to compute more abstract properties, such as edges or connectivity between objects. In the pedestal blocks world example in Figure 5, the agent imagines the movement of the blocks, creating a persistent state in the imagery system to which it applies high-level perception to infer that moving **B** to **`pedestal2`** will cause a collision. However, one could create a similar system where the same information (future collisions) is calculated via the same concrete routines (geometric operations in the concrete representation), but where the abstract decision system does not access it by applying perception to an imagined state. In that case, a predicate such as **`collision_if_moved(A,peg1)`** might simply be provided by high-level perception, which happens to use concrete routines to determine it.

[7] Concrete routines are a concept that can apply to any concrete representation, not just visual. Another difference from Ullman's theory is that here, imagery is a specialization of concrete routines, where Ullman proposed visual routines as a means to perception of abstract properties, not as imagery operations. Pinker, however, posited a commonality between visual routines and imagery (Pinker, 1984).

Figure 7(e) shows a system that encodes both concrete and abstract representations, but does not use architectural mechanisms to convert between them. That is, the process of converting between one format and another is left to knowledge, or the system may not differentiate between knowledge and architecture. For example, an agent mixing quantitative and abstract information in a common architectural memory with knowledge that converts between the formats meets this description. This system might have all of the other aspects in it, but they cannot be seen in the diagram because they are not supported by architecture.

Finally, Figure 7(f) shows an architecture where the imagery system is encapsulated in a module, and not connected to perception and action. Again, this architecture might have every other aspect in it, but imagery processing here is distinct from perception and action as it relates to the external world.

Reasons for preferring the proposed architecture over these alternatives will be more fully discussed in Section 3.3, which covers the functional benefits of the theory.

3.2 High-level Claims and Evaluation Approach

As has been stated, the central goal of this work is to investigate cognitive architectural mechanisms to support intelligence in spatial tasks. The central claim is that the theory outlined in the previous section makes progress towards this goal. This progress can be captured by three high-level claims:

- The theory allows for improved performance in individual spatial tasks.
- The theory allows for improved generality: more tasks can be covered.
- The theory is practically useful: an architecture following the theory can be implemented and used.

To show evidence for these claims, in the next section, specific benefits afforded by the architecture are introduced. Each of these benefits relies on a certain subset of the architectural aspects outlined in the previous section, and each supports one or more of the high-level claims. For those benefits supporting the first two claims, the

performance or generality of an architecture following the theory can be compared to that of an architecture that is otherwise the same, but lacking the theoretical aspect(s) supporting the benefit. This comparison is the basis for arguing "improvement" as stated in the claims.

In later chapters, a comprehensive implemented architecture is presented, and agents instantiated in that architecture are demonstrated. These demonstrations serve to directly evaluate the third claim. While evaluation of the first two claims is supported by evidence from these demonstrations and experiments, it is important to note that those claims cannot be directly evaluated with experimental data. There are many free parameters in the design of an architecture and agent if the only constraints on it are that it is symbolic and lacks a particular aspect. This makes it extremely difficult to develop alternative agents that can be used in truly fair experimental comparisons to agents following the complete theory. Instead, while implemented agents and empirical data are presented to demonstrate the benefits, the evaluation of the first two claims is primarily based on analysis as opposed to empirical results.

3.3 Benefits of the Theory

In this section, specific functional benefits of the theory are described. Outside of the implementation of the architecture itself (as described in the next chapter), a thorough exploration of the functional benefits of simulative imagery and the other architectural aspects is a major contribution of this work.

As discussed in the previous section, the three main claims here are that the architecture allows the agent to solve particular tasks better compared to alternatives (*task performance* is improved), that the architecture can address more tasks compared to alternatives (*generality* is improved), and that the architecture can be usefully implemented. Broadly speaking, the particular benefits outlined here can be categorized based on which of these goals they further the most: they are task performance, generality, or design benefits (those which simplify the implementation).

Improved generality and task performance are tightly intertwined goals: improving task performance can aid generality if the task improvement moves the architecture from being unable to address a problem (by meeting some minimal standard of behavior) to being able to address it. In addition, an improvement in generality is also an improvement in task performance, if one simply defines the "task" being addressed as solving some set of primitive tasks. Here, we restrict generality benefits to those that are most applicable to what are conventionally called task-independent systems.

At a higher level, for some benefits, the means by which task performance or generality is improved is by directly addressing the perceptual abstraction or irreducibility problems. When a benefit can be understood in these terms, it is noted.

The individual benefits are described in detail below. Each is supported by particular theoretical aspects, as is described. For task performance and generality benefits, the improvement claimed is compared to a similar architecture, but lacking the necessary aspect(s) supporting the benefit.

For convenience, Tables 1 and 2, located at the end of the chapter, summarize the theoretical aspects of the architecture, the benefits, and the associations between them. Table 2 also references agents that are introduced in later chapters that demonstrate the benefits.

B1. Concrete routines allow movement and nonlocal interaction to be captured in terms of abstract symbolic information, mitigating the perceptual abstraction problem.

In order to choose an action, an agent may need to take into account the precise movement of objects. For example, when parking in a parking garage, one needs to consider whether the car will collide with a pillar when turning into a tight spot. Similarly, it can be necessary to take into account object interactions that, from the perspective of the current state, are non-local. The pedestal blocks world again provides an example of this: when considering moving a block from the bin to a pedestal, it must

be determined whether a collision will occur in the future. In the current state (when the block is on the table), this determination involves both properties of the moving block and properties of objects that are spatially distal from it.

To represent these problems at an abstract level, the perception system must distinguish between the relevant states. Even if the agent has a perception system specifically built for the task, though, this is a difficult perceptual abstraction problem if concrete routines are not present. The distinctions cannot be easily detected by composing simple "features" of the current visual scene detected in a bottom-up manner, or by matching the scene to memories.

However, if actions can be simulated based on concrete information, properties that were difficult to compute in the original state might be simpler to compute in the simulated state. In the pedestal blocks world example, once the block is imagined in its new position, the agent need only infer a basic property: whether or not the block collides with any other object. Using simulation, the necessary properties can be inferred, improving task performance.

The minimal aspect of the architecture necessary for this benefit is concrete routines (A2). Note that *imagery* is not necessary; a system could leverage this benefit without having the decision system deliberately invoke simulations, and without having persistent resulting concrete states. For example, to address pedestal blocks world, the perception system of an imagery-less architecture like that in Figure 7(d) could provide a predicate like **`collision-if-moved(A,pedestal2)`** as a result of an automatic concrete simulation, capturing a non-local interaction. The next benefit explains why imagery might be a preferable approach, though.

Concrete routines also allow for a system to encode useful properties beyond movement and nonlocal interaction, such as object connectivity (Ullman, 1984). Concrete routines supporting these properties may not simulate actions, as the routines that have been discussed so far do.

Since relevant properties can be easily detected by a system with concrete routines that cannot be easily detected by a system without, aspect A2 supports improved task performance via this benefit.

B2. Imagery allows task-specific abstract properties to be encoded by a fixed, task-independent high-level perception system, mitigating the general perceptual abstraction problem.

This benefit addresses the general perceptual abstraction problem. How can a task-independent agent construct abstract perceptual properties in arbitrary tasks? This is a hard problem, since deriving abstract properties from concrete information is a difficult process. The simplest approach to this would be to come up with a set of universal abstract properties, which are calculated by architectural means.

However, for spatial problems, this approach does not seem viable. Researchers in qualitative spatial reasoning have attempted to describe such a set of universal properties, but no such set has been found. This has led to the poverty conjecture of Forbus et al. (1991):

> "We claim there is no purely qualitative, general-purpose, representation of spatial properties. That is, while qualitative descriptions are useful in spatial reasoning, they are not sufficient to describe a situation in a task-independent and problem-independent fashion."

Task-independent qualitative properties are precisely the sort of abstract symbolic representation of perceptual information that allow an agent to compactly represent the state of a problem while retaining enough information to choose appropriate actions. Assuming the poverty conjecture is true, something more is needed to solve the general perceptual abstraction problem.

For solving qualitative reasoning problems, Forbus et al. propose augmenting qualitative information with a quantitative representation, which is similar to the approach taken here. If imagery is present, the overall process of perceptual abstraction can involve

both concrete manipulation and high-level perception. From an architectural point of view, the same high-level perceptual processes allow different symbolic information to be encoded depending on how the concrete representation has been manipulated. In this way, if the set of concrete manipulation processes change or are applied in different ways in different tasks, the process of perceptual abstraction is more flexible. Not only can more relevant properties in particular tasks be generated (as the previous benefit covered), but this flexibility allows the architecture to encode different task-specific properties in different tasks, improving generality.

As in the pegged blocks world example, in this work we focus on using simulative imagery (A4) to generate these task-specific properties. In that example, the fact that a collision results from a particular action is a (task-specific) property of the current state. Non-simulative imagery could be used for the same purpose, though. For example, an agent could use geometric imagery operations such as creating a line between two objects to determine if a third object is between them. However, it is important to note that imagery (A3), and not concrete routines alone (A2), provide this benefit. That is, it is important that the concrete routines are selected and controlled by the abstract decision system, and that the resulting state is persistent. To allow the agent to adapt to new tasks, the full reasoning power of the agent must be brought to bear to select which concrete routines to apply and how to interpret or further manipulate the results. An architecture like that in Figure 7(d) cannot support this, as the process of selecting routines and interpreting results is isolated within the perception system and disconnected from the agent's general-purpose knowledge, which, we are assuming, resides in the abstract decision system.

Since imagery (A3) allows an agent to encode task-specific properties with a task-independent perception system (and the poverty conjecture appears true), this aspect can result in improved architectural generality.

B3. Simulative imagery provides the agent with the ability to abstractly model actions that are non-deterministic at the abstract level.

Action models have often been used in AI systems as a means to inform action choices and improve task performance. For example, using an action model, an agent can internally search through a problem space to determine an action sequence that leads to a goal (Newell, 1990). Similarly, a model-based RL agent uses an action model to internally simulate experience in order to learn a policy (Sutton & Barto, 1997). Essentially, both of these techniques are processes of taking knowledge of how actions affect state transitions and "unrolling it" to make explicit knowledge about which actions to take. As simulative imagery supports action modeling, it similarly allows this benefit.

However, while simulative imagery can be used for action modeling, due to its use of multiple levels of abstraction, an imagery system is not simply equivalent to a set of action models. A standard action model produces predictions of future states (or state changes) based on the current state and proposed action. Importantly, both the states that the predictions are based on and the states the predictions produce are from the same state space. This is not the case with imagery, as predictions are based on concrete states, but produce both concrete states (in the imagery system) and abstract states (in the decision system). This difference is clear in the pedestal blocks world example (Figure 5). Given a single abstract state and action, the system could predict different successor states (collision or non-collision states) depending on details that are not encoded in the original abstract state. From the perspective of the decision system, the predictions are non-deterministic.

Predictions of this sort can be viewed as both action models and inferences about the current state. A prediction that a collision will occur given a particular action both distinguishes the state from one in which the collision will not occur (as discussed in B1) and predicts a future state. However, action models can be "unrolled" to achieve performance benefits above and beyond what is possible if the prediction is simply

assumed to be an inference about the current state, which sets this benefit apart from B1.

Due to abstract non-determinism, difficulties may be encountered using techniques like problem space search or model-based RL with simulative imagery action models. If the predictions of imagery are non-deterministic in terms of abstract states (as in the pedestal blocks world), dynamic programming techniques, including chunking in Soar, cannot be used. If the agent encounters an abstract state it has previously examined, that state does not contain sufficient information for the agent to know whether the result of previous processing still applies. Nevertheless, simulative imagery can still provide a benefit of action modeling, even in situations where dynamic programming is impossible, as will be demonstrated in later sections.

Simulative imagery (A4), rather than concrete routines (A3), are necessary for this benefit. If an agent is to model its actions, the decision system needs to be able to choose which action to model, and the state must be persistent in case further actions are to be simulated. With concrete routines alone, these capabilities would not be present. Since simulative imagery can allow action modeling even when abstract state transitions may appear nondeterministic, improved task performance is possible.

B4. Simulative imagery allows decisions to be made using an abstract representation while predictions are made using a concrete representation, allowing each process to use the representation that allows the most efficiency.

When making decisions, abstraction can be a great benefit. The level of abstraction in a reinforcement learning agent's state space affects the speed at which it will learn, and for any agent that uses symbolic knowledge to choose actions, the degree of state abstraction influences the amount of knowledge necessary to solve a problem.

Conversely, when making predictions, abstraction is not always beneficial. The movement of objects in the physical world can often be characterized with simple equations that operate over quantitative, concrete information. Over long time scales,

this form of prediction is increasingly difficult and inaccurate, and qualitative prediction (e.g., Forbus, 1983) may be a better approach, but over short time scales, quantitative prediction can be very simple and accurate.

Similarly, in problems like the blocks world, the consequences of a movement can be easily described and simulated in concrete geometric terms—the block will be positioned such that it will be centered on top of the object below. Action modeling in problems like the blocks world can be performed either concretely or abstractly, but concrete simulation eliminates such difficulties as encoding the relevant frame axioms for the transition, as has been argued by others (Huffman & Laird, 1992; Glasgow, 1995; Kurup & Chandrasekaran, 2006).

Allowing this flexibility in the means of implementing action models can improve task performance, and is supported by simulative imagery (A4) for the same reasons B3 is.

B5. Concrete control allows continuous control processes to be used in conjunction with abstract symbolic reasoning, mitigating the irreducibility problem.

In the previous chapter, it was argued that irreducibility is a problem even in simple tasks like those in the blocks world. An agent must be able to adapt its outputs to tiny variations in the state of the world in order to perform an action like grasping a block. In many problems, this issue can be addressed by incorporating low-level controllers in the system, and dividing the task between a decision system that chooses between which controller to invoke, and the controllers themselves, which actually execute the action. The controllers have access to much more precise state information than is used to make decisions. In that way, a blocks world problem can be abstractly solved by a robot through reasoning about grasping a block, moving a grasped block, etc., and the controllers to perform those actions can vary their output to exactly conform to the state of the world (Laird et al., 1991).

From the perspective of the decision system, concrete controllers change the action space of the task: where raw motor outputs would be selected before, now the agent

can select between controllers or parameterizations of controllers (e.g., “reach for block A” or “reach for block B” rather than “set motor 2 to .02 volts”). Given the transformed action space, perceptions can also be transformed and made more abstract, since the number of states the decision system needs to distinguish between to select a controller is likely to be smaller than the number of states it would need to distinguish between to select a raw motor output.

This division of labor is supported in the proposed architecture via aspect A5, concrete control. It should be noted that the theory implies that all controllers base their actions on a common concrete representation. It remains to be seen whether a common set of concrete information can encapsulate everything any controller in a human-level system may need to know to determine its output, or if individual controllers will need perceptual information beyond what is in the concrete representation.

Since concrete control can reduce the effective state space of a problem, allowing for more efficient decision making, task performance can be improved.

B6. Simulative imagery of concrete control allows symbolic reasoning over continuous processes, eliminating the need for symbolic characterization of controller performance, further mitigating the irreducibility problem.

While irreducibility in some problems can be handled entirely by encapsulating appropriate behaviors in concrete controllers, this is not always the case. As with the motion planning example in the last chapter, sometimes controllers cannot be built such that the problem can be reduced to an abstract state space. However, if simulative imagery of concrete control (A6) is present, abstract symbolic processing in the agent can reason over controllers without having a characterization of their behavior in terms of abstract states. The simulation allows the agent to derive the abstract outcome of a proposed action in the particular situation, even though that outcome might depend on details of the situation that the agent cannot capture in terms of abstract symbols.

This allows for much less constraint on the kinds of controllers that can be used in the system. Performance is then improved in tasks like nonholonomic motion planning, where the problem cannot be otherwise addressed without some loss of solution quality.

B7. Architectural representation conversion encapsulates complex, common processes, rather than requiring task-specific knowledge.

As an alternative to the theory proposed here, a system could be constructed that does not include fixed architectural processes to support representation conversion (Figure 7(e)). Conversions between representational formats (high-level perception and imagery) would be performed by task knowledge. This results in an alternative way of approaching the general perceptual abstraction problem—if no fixed processes are present to build abstract representations, the problem of building appropriate task-specific representations is left up to the agent designer (or the agent itself), rather than the architecture designer.

However, this alternative seems infeasible. Converting between representational formats can be very difficult, and it is not clear what sort of system would be needed to learn to do this conversion. In addition, many forms of representation conversion are useful across many tasks. For example, high-level perception in both the pegged blocks world and motion planning examples detects collisions between objects, so a generic high-level perception system that can encode abstract information about object collisions in the concrete representation is useful in both tasks.

This commonality across tasks is present in the action system, but to a lesser degree. It is reasonable to assume that an intelligent agent must learn new low-level controllers to address new tasks throughout its life, so at least some knowledge must go in the action system. All controllers can use the same architectural interface to the decision system, despite their low-level differences, however.

Since complex, common processes are in the architecture, rather than encoded as knowledge, they need not be learned or re-engineered in new tasks, improving generality. However, it is an open question as to what set of high-level perception and action processes exists that can be implemented to support human-level cognition. In the next chapter, a rough proposal for these architectural mechanisms will be put forth.

In addition, architectural processes can be much more efficient than equivalent processes encoded in as task-independent knowledge (Lathrop, 2008). For this reason, this is a task performance benefit in addition to a generality benefit.

B8. Perception/action reuse provides for a parsimonious set of architectural processes.

The final architectural aspect, perception and action reuse, has a straightforward associated benefit. In an architecture where perception and action systems are not reused for imagery (Figure 7(d)), agent performance may not be substantially affected, but the design of the architecture itself would be complicated. In general, the abstract information that is useful to extract from imagined states is similar to that which is useful to extract from direct perception, so the same mechanisms can be used. A concrete representation is useful as an intermediate construct during perception (Ullman, 1984), so there is no compelling reason to use an isolated imagery representation. And the same low-level controllers useful for external actions can be used internally to simulate those actions (e.g., Leonard et al., 2008), so again, there is no reason for isolated systems.

Since there does not seem to be a functional benefit for separate imagery and perception/action processing, there is a strong functional benefit to be gained in terms of architectural simplicity by unifying them.

Architectural aspects	Pre-requisites
A1 Bimodality	-
A2 Concrete routines	A1
A3 Imagery	A1,2
A4 Simulative Imagery	A1,2,3
A5 Concrete controllers	A1
A6 Simulative imagery of concrete control	A1,2,3, 4,5
A7 Architectural representation conversion	A1
A8 Perception and action reuse	A1,7

Table 1: Aspects of the theory

Benefits	Supporting aspect	Main high-level benefit	Demon-stration(s)
B1 Symbolic capture of movement and non-local interaction	Concrete routines (A2)	task performance (perceptual abstraction)	all agents
B2 Task-specific abstract property generation	Imagery (A3)	generality (perceptual abstraction)	all agents (collectively)
B3 Abstract action modeling	Simulative imagery (A4)	task performance	all agents
B4 Abstract decisions with concrete predictions	Simulative imagery (A4)	task performance	all agents
B5 Abstract decisions with concrete control	Concrete controllers (A5)	task performance (irreducibility)	RRT agent
B6 Symbolic reasoning over continuous control	Simulative imagery of concrete control (A6)	task performance (irreducibility)	RRT agent
B7 Complex, common processes in architecture	Architectural representation conversion (A7)	generality, task performance	all agents, architecture design
B8 Architectural parsimony	Perception and action reuse (A8)	design	all agents, architecture design

Table 2: Benefits of the theory, mapping to aspects and demonstrations

Tables 1 and 2 summarize the theoretical aspects of the architecture, the benefits, and the associations between them.

Chapter IV - The Soar/SVS architecture

In this chapter, I outline the design and capabilities of the Spatial/Visual System, or SVS, which together with the existing Soar architecture (Laird, 2008) constitutes an implementation of the theory presented in Chapter III. SVS inherits from two previously-separate projects, Spatial Reasoning for Soar (SRS), which I developed (Wintermute & Laird, 2007, 2008), and Soar Visual Imagery (SVI), which Scott Lathrop developed (Lathrop, 2008; Lathrop & Laird, 2007, 2009). The relationship of SVS to these prior systems is discussed in detail in Section 7.2.

In general, the discussion here is at a more detailed level than in the previous chapters. Planning, reinforcement learning, simulative imagery, etc., are different techniques that an agent instantiated in Soar/SVS can use to solve problems, but the architecture itself can support many other techniques. More specifically, SVS contains parts that can be composed together to support simulative imagery, it is not simply a simulative imagery module—the parts can be alternatively used to support very different methods. This is part of the overall goal of working towards an architecture capable of supporting general spatial intelligence. Simulative imagery is one important method that can enhance spatial capabilities in many problems, but it is important that the architecture itself be posed in as general terms as possible, so that other methods can also be used.

4.1 Overview

The overall design of SVS is shown in Figure 8. This diagram is decomposed in a similar way to Figure 6(b), in terms of a decision and imagery system, and the connections between them. Soar is the decision system in this case[8]. Agents in Soar can be instantiated to use many different techniques to make decisions, including planning and

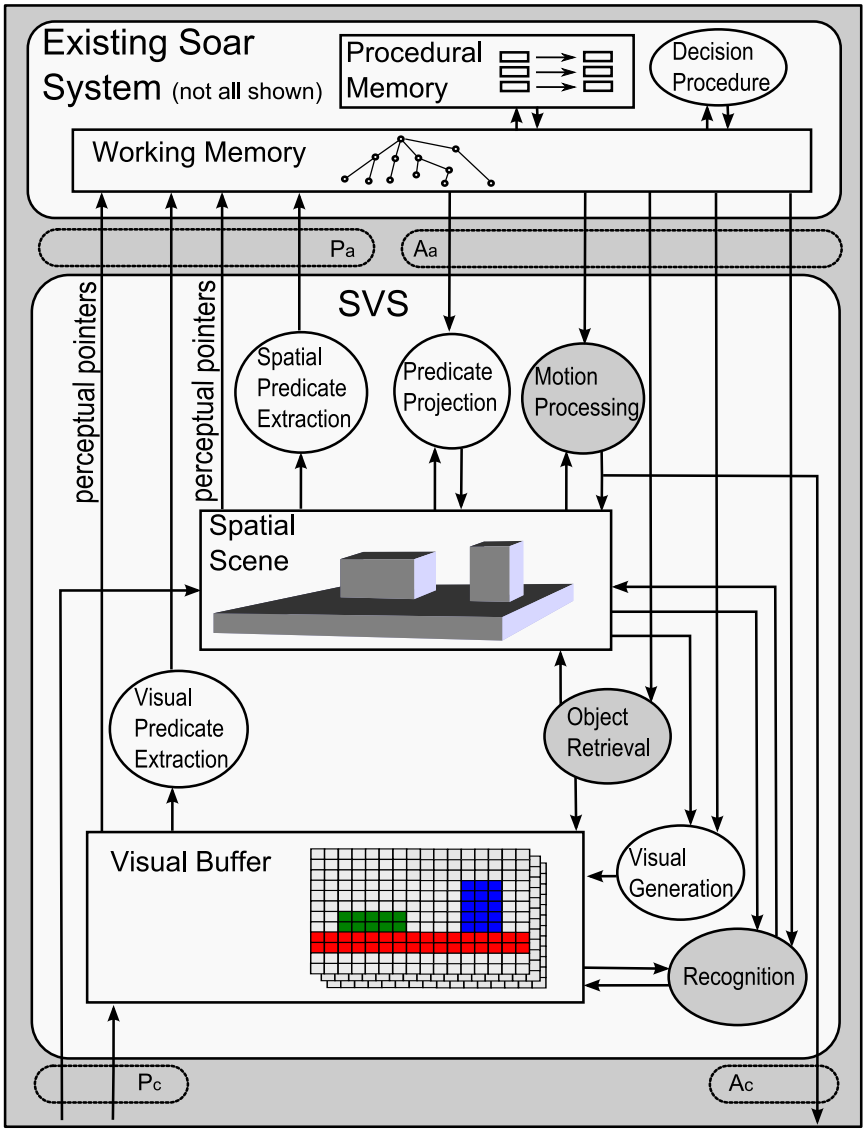

Figure 8: SVS System design. Boxes are short-term memories, circles are processes. Grey circles involve access to information in long-term memory (knowledge). There are implicit control lines (not shown as arrows) between working memory and all of the processes shown.

[8] SVS is an extension to the Soar architecture, however, in this document I use "Soar" to refer to the previously existing components of Soar, and "Soar/SVS" to refer to the combined system.

reinforcement learning. Soar contains a symbolic working memory, through which different processes in memories in Soar communicate. This is where SVS connects to the existing Soar system, high-level perception (via $\boldsymbol{P}_a$) adds elements to a special area of working memory, and high-level actions (issued via $\boldsymbol{A}_a$) are similarly formulated in a special area of working memory. The $\boldsymbol{P}_a$ and $\boldsymbol{A}_a$ signals have many meaningful components (there are many "forms" of perception and action), as shown in the diagram, and is explained later. Other cognitive architectures, such as ACT-R (Anderson et al., 2004), similarly use symbolic structures to connect to the outside world, and the concepts behind SVS (if not the software itself) could be adapted to any system with such an interface.

As an imagery system, SVS sits between symbolic processing in Soar and the outside world. A complete embodied agent also requires lower-level perception and action systems to handle the actual output of sensors and input to effectors. These systems are the source of the $\boldsymbol{P}_c$ signals and receiver of the $\boldsymbol{A}_c$ signals, respectively.

The memories inside SVS are influenced by Kosslyn's theory of visual imagery (Kosslyn et al., 2006). There are two short-term memories in the system, for spatial and visual information. The visual buffer represents information in 2D arrays of pixels, analogous to the part of the human visual system active during classical imagery tasks. This represents strictly visual information, including precise shapes and colors. In contrast, the spatial scene contains 3D information that is (theoretically) derived from multiple senses, quantitatively represented as continuous coordinates describing polyhedrons in 3D space[9].

Both of these representations are concrete representations—they make as many distinctions between external states of the world as possible. In this work, however, the visual system is not be used, although it is a part of the architecture and its components

[9] This is the equivalent of the Object Map in (Kosslyn et al., 2006), and in (Lathrop, 2008), which inherits the terminology.

will be briefly discussed in this chapter. Lathrop (2008) focused on usage of the visual system in his work on a predecessor of this architecture.

In addition to the two short-term memories, there is a long-term memory in SVS for visual, spatial, and motion data, called perceptual LTM. To simplify the diagram, this memory is not explicitly shown, but is accessed by the object retrieval and motion processes. While these memories contain chiefly non-symbolic information, they also are partly symbolic—unique objects are given identifying symbols, through which the symbolic aspects of the system can refer to them. These identifying symbols, here called *perceptual pointers*, are similar to the visual indexes described by Pylyshyn (2001) for short-term visual memory, in that the system "... picks out a small number of individuals, keeps track of them, and provides a means by which the cognitive system can further examine them in order to encode their properties ... or to carry out a motor command in relation to them".

For real-world agents, perception is a major challenge. Theoretically, the perceptions provided to SVS ($\boldsymbol{P}_c$) should be raw pixels from a camera, or something analogous to the lowest cohesive representation in the human visual system. This is the component of $\boldsymbol{P}_c$ that feeds directly to the visual buffer. Theoretically, memories and processes inside SVS, with influence from symbolic processing in Soar, should segment and recognize objects and estimate 3D spatial structure based on 2D visual information. As we do not address the veridical perception problem, the system does not attempt this. However, limited capabilities are present to recognize objects in the visual system, carried over from SVI's ability to manipulate the visual representation to identify new objects or object features (e.g., the enclosed space in the letter A).

However, for many virtual environments (and some limited problems in real environments), it is possible for a separate perception system to provide information at a high-enough level that SVS can be used without a complete visual system. Many simulated environments directly represent the world as labeled 3D polyhedrons; in that case, those objects and labels can be directly fed into the spatial scene via $\boldsymbol{P}_c$. In the

current implementation, the $\boldsymbol{P}_c$ component to the visual buffer is not used, only the component to the spatial scene.

From the point of view of the decision system, the only aspects of the underlying visual and spatial state available are what are encoded in $\boldsymbol{P}_a$. Here, that includes the perceptual pointers, along with information available through spatial and visual predicate extraction[10] processes. These processes, outlined fully in the next section, extract qualitative symbolic information from the underlying quantitative state. For example, the agent can detect whether or not two objects intersect. Note that these processes do not involve access to knowledge: there is a fixed, architectural library of predicates that the system can extract. The exact visual and spatial details of the objects in the world (their coordinates in 3D space) are *not* provided in $\boldsymbol{P}_a$.

Since the information available to the symbolic system is limited to object identities and simple qualitative properties, for complex reasoning tasks, imagery must be used. To perform imagery, the system needs mechanisms through which visual and spatial images can be created and manipulated. In SVS, there are four such mechanisms, which will be outlined in the following sections. These mechanisms include memory retrieval, which instantiates objects from long-term memory into either the visual or spatial STM, motion simulation, which moves images in the spatial scene in the same way a motor action or other motion in the world would, and predicate projection, which creates spatial objects based on qualitative descriptions created by Soar (such as "a line between **A** and **B**").

4.2 Perceptual Pointers

The most basic form of information passed between SVS and symbolic processing in Soar is the perceptual pointer. A perceptual pointer is a unique token, which refers to a

[10] This terminology is inherited from work in diagrammatic reasoning (Chandrasekaran, 1997). The term "predicate" is perhaps overly formal, since it might imply that predicate logic is being used for inference in the system, which is not the case, but it has the correct implication that we are dealing with symbolic properties of objects.

specific underlying visual or spatial structure. A pointer appears as an **id** attribute in Soar's working memory. In addition, if a structure is recognized as an instantiation of a structure in perceptual LTM, it is augmented with a **class-id**.

The types of structures that use this identifier system are spatial objects, spatial transformations, visual textures, and motion models, all of which are discussed below. The perceptual pointer provides a simple means by which a symbolic working memory structure can refer to an underlying perceptual structure: the pointer id is generated by SVS, and every time symbolic processing in Soar uses that id in a context SVS understands (e.g., an imagery specification to "imagine **car23** to the right of **house12**"), SVS uses that id to access the underlying perceptual structure (e.g., the polyhedron describing the car) from its internal memories.

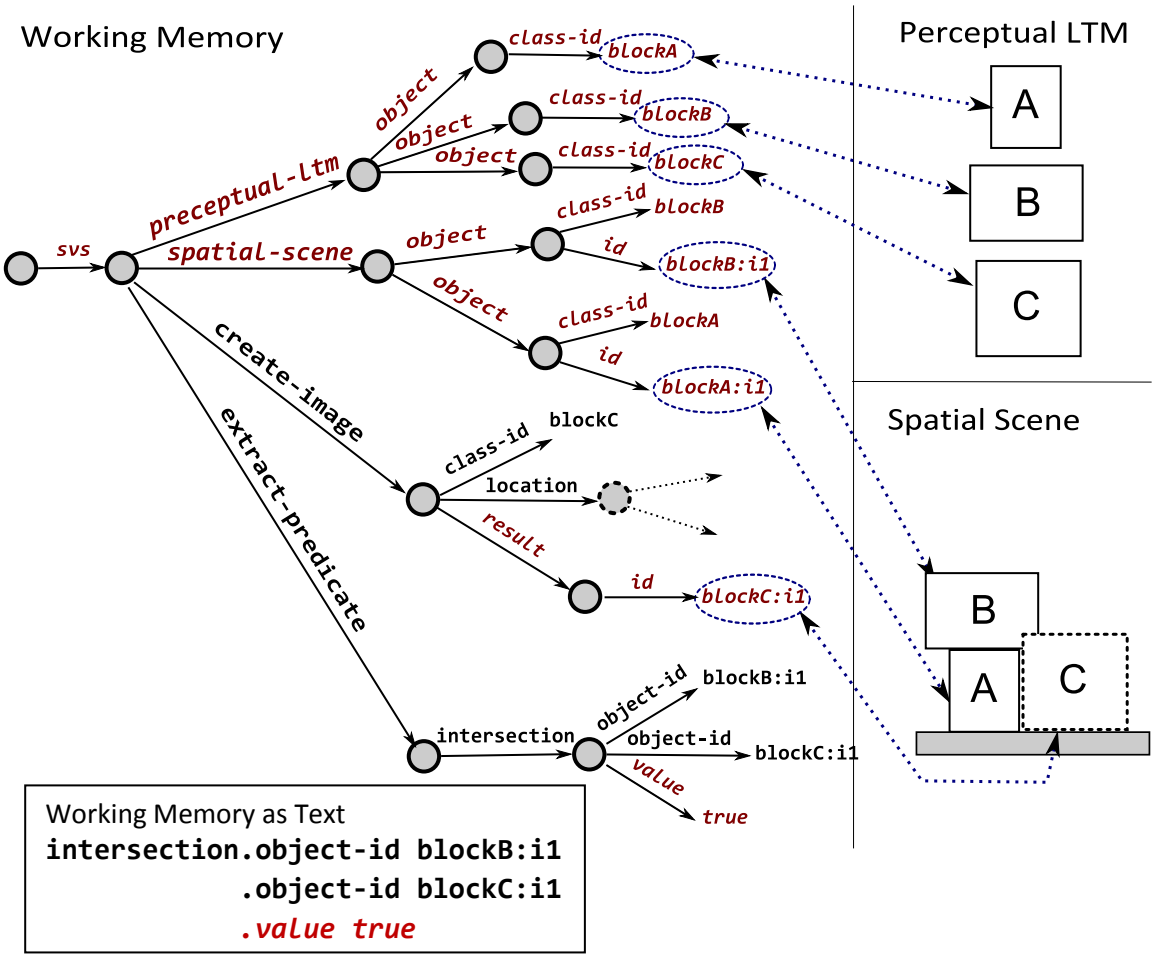

Figure 9: Top, blocks world information in Working Memory, Perceptual LTM, and Spatial Scene. Bottom, equivalent text representation of the extract-predicate structure. Working memory structures in *red italics* are created by SVS.

Figure 9 shows blocks world information represented in Soar/SVS. Soar's working memory is shown on the left. Structures in working memory are represented by a directed graph of symbols that are matched and manipulated by rules. A portion of this graph is shown in the figure. Working memory structures can also be represented with text, as shown.

Dotted arrows in the figure show perceptual pointers, represented in working memory as symbols. Only those created by SVS have arrows in the figure, but other instances of the same symbols are also pointers to the same objects.

4.3 Spatial Scene Encoding

The spatial scene is SVS's short-term memory for spatial information. It normally contains the structure of the world around the agent (including parts it cannot immediately see), or the structure of an imagined situation, or, more commonly, a mixture of both. Internally, the scene is a set of 3D polyhedrons grounded in continuous coordinates. This information is presented to Soar's working memory as a hierarchically-organized tree of objects, with the tree structure indicating part-of relationships. Each object node includes a perceptual pointer by which Soar can refer to it. Between each object node, a transformation node is present. Each transformation node contains a perceptual pointer to the relationship between the two objects.

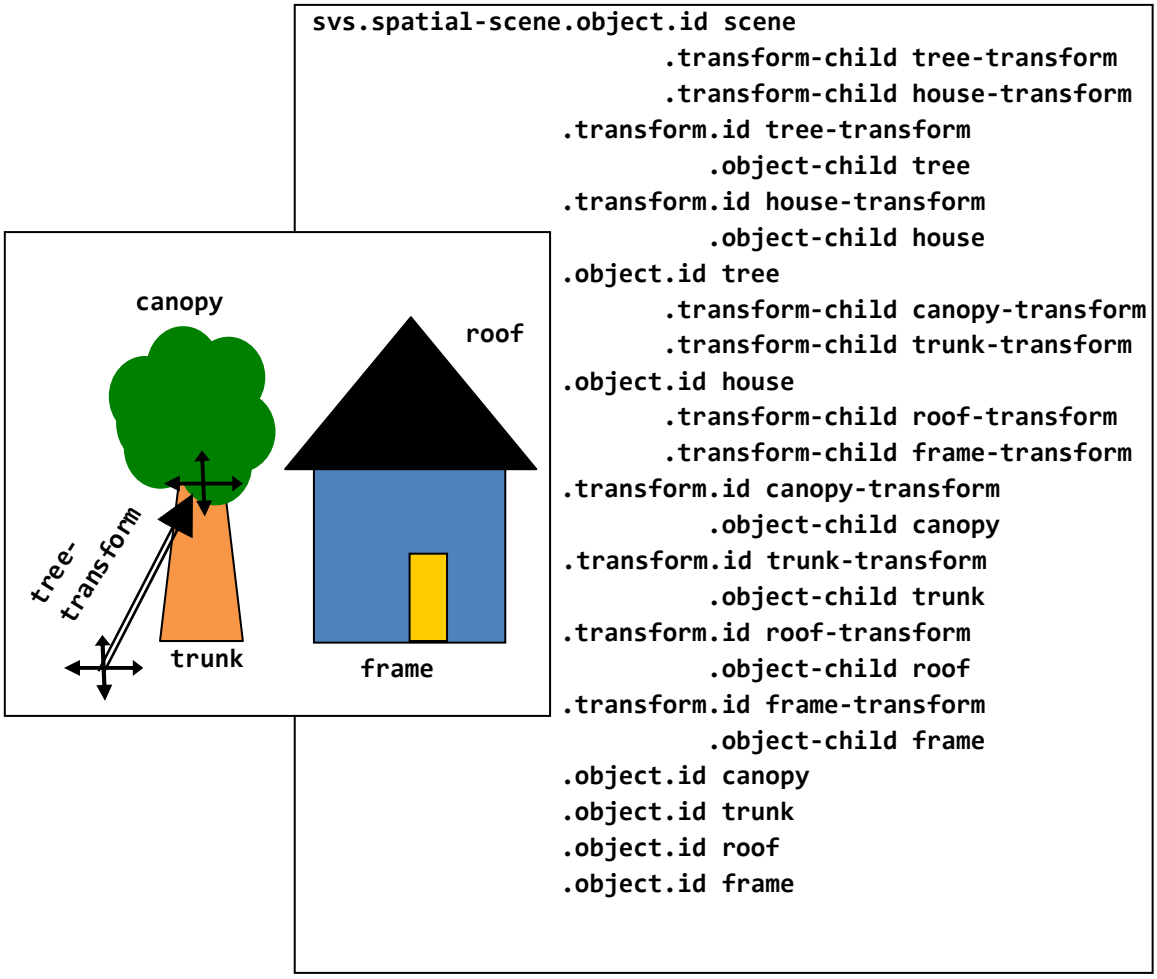

Figure 10: An example of the symbolic encoding of the spatial scene in Soar.

Objects in pegged blocks world lack hierarchical structure as would be present in more complicated environments, so, for simplicity, Figure 9 shows only the object nodes at one level of the hierarchy. Figure 10 shows a scene with richer structure, and the full detail of how the spatial scene is encoded (with the exception of **`class-id`**s). Only the leaves of the tree in working memory correspond to primitive polyhedrons (which are labeled in the Figure), but nodes at every level are considered objects. Everything in the scene is part of one **`scene`** object (the root of the tree), and the **`scene`** object has children for its parts, the **`tree`** and **`house`** (connected via their transformations), which are further decomposed to primitive parts. Much of the processing in SVS (e.g., detecting an intersection between two objects) relies on the assumption that objects

are convex, so when objects are referred to, what is actually used is the convex hull of all of the parts below that object. If non-convex objects are to be used, the environment must encode them as a set of convex parts, each of which the system can reason with independently. Automating this convex decomposition is an area for future work.

Transformation nodes exist between each object node in the hierarchy. Each object has some intrinsic reference frame, and the transformation node refers to the relationship between those frames. The figure shows one of those transformations, **`tree-transform`**, which relates the reference frame of the **`scene`** object to that of the **`tree`** object.

It should be emphasized that the perceptual pointers that make up the scene graph in Soar's working memory refer to the actual objects and transformations in the spatial scene. If the agent refers to the **`id`** of a transformation in the spatial scene, it is referring to a specific quantitative transformation in 3D space, *not* a generic qualitative relationship like "towards the upper right" of "northeast of". If the agent retrieves a transformation from perceptual long-term memory (by referring to its **`class-id`**), that transformation is similarly quantitative. In general, everything in perceptual LTM and in the STMs of SVS is quantitative. However, symbolic processing in Soar can only *refer* to quantitative perceptual information, it cannot directly access the actual quantities involved. Rather, it accesses that information only indirectly, by querying the scene for qualitative information through the predicate extraction process (to be covered shortly).

4.4 Predicate Extraction

The predicate extraction processes serve to provide symbolic processing in Soar with qualitative properties of the contents of the spatial short-term memory in SVS. These processes are fixed parts of the architecture; there are no plans to enable new forms of predicate extraction to be learned by the agent itself. In contrast to perceptual pointers, qualitative predicates are created in working memory only when requested by processing in Soar. There is a great deal of qualitative information implicit in the memories of SVS, each piece of which can take substantial calculation to derive, so

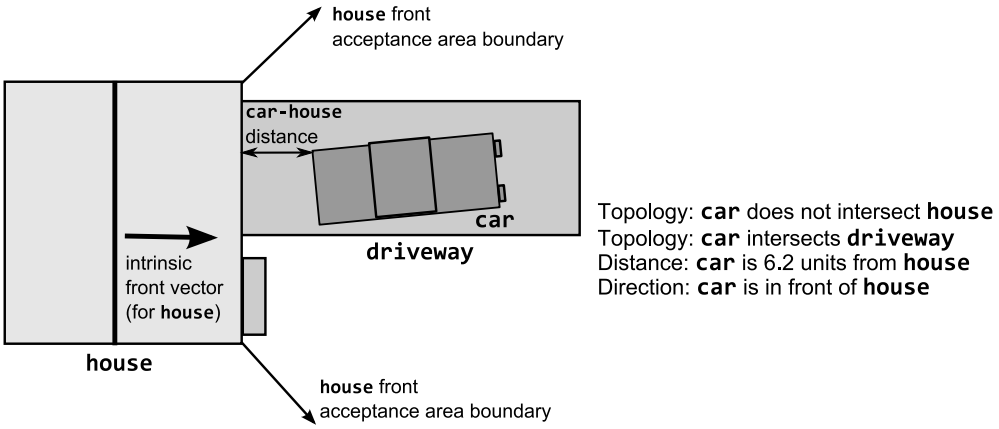

Figure 11: Information derivable through spatial predicate extraction.

some attention mechanism is needed to make the system computationally tractable. The process of requesting predicate extraction from working memory is called *querying* SVS. Figure 9 shows an example of predicate extraction in working memory. First, symbolic processing in Soar creates a structure describing the query (in the example, whether or not two blocks intersect) in the proper area of working memory. SVS then recognizes the query structure, calculates the desired property (in this case, detecting the collision), and builds a result structure in working memory. Further symbolic processing in Soar can recognize this result, and reasoning will continue.

For the spatial system, there are three important kinds of relationships between objects that might be queried for: topology, direction, and distance. An example of each of these relationships is illustrated in Figure 11. Topological relationships describe how the surfaces of objects relate to one another. Work in qualitative spatial reasoning has explored these relationships (Cohn et al., 1997). While schemes exist that represent several possible cases of topological interaction (e.g., discrete, partially overlapping, proper part), the only topological information currently available in SVS is whether or not two objects are intersecting or not (that is, colliding). More relationships might be added in the future, but no tasks have yet been addressed that require more detailed topological information.

Distance is similarly simple, currently the system is able to query for the distance between any two objects in the scene, along the closest line connecting them. In the

makes the information arguably non-qualitative, although it is certainly "less quantitative" than the contents of the spatial scene, as it reduces complex three-dimensional information to a scalar quantity. However, it is extremely useful in practice. The common use for distance information is simple distance comparison, which can be done easily with Soar's existing number-comparison functionality. The closest obstacle to the agent might be detected by extracting the distance from the agent to all of the obstacles, and comparing the distances to determine the closest. If we wanted to strictly evict all continuous numbers from Soar's symbolic processing, this could be replaced with a "closest" extractor, but that approach seems to be unnecessarily complicated[11].

To support orientation relationships between objects, and determine information such as "object A is to the left of object B", a final class of orientation queries is implemented. Following the approach of Hernandez (1994), for each object, a set of surrounding *acceptance areas* is defined. An acceptance area corresponds to a region of the world where all points in that region share a common orientation with the object in question. These regions roughly correspond to concepts like left, right, front, back, above, below, etc. An example of an acceptance region is shown in Figure 11, where the region considered to be in front of the house is indicated. All acceptance areas are calculated relative to a bounding box around the object, queries for relationships inside of that box are undefined. Three different methods of constructing acceptance areas are possible. As shown in the figure, standard orientation queries construct the acceptance area with bounding rays emanating at 45-degree angles from the corners of the object. In contrast, a *strict* orientation query would project the rays in the figure directly to the right, making the "front" region of the house smaller, and a *general* orientation query would project the rays up and down, making the region larger.

All objects have an intrinsic frame of reference, and orientation queries are often performed relative to that frame. However, orientation queries can be performed in the

[11] The use of continuous numbers as part of Soar's working memory does not have strong theoretical support, but as implemented, they provide practical benefits. Issues related to SVS and the role of numbers in Soar's working memory are discussed in the Appendix.

frame of reference of any object in the scene[12]: that is, the frame of reference of a different object can be used to define the acceptance areas. In this way, allocentric orientations can be extracted by using the frame of reference of the entire scene (or of a compass object in the scene), and interpreting the results as cardinal directions (North, South, East, West, up, down).

For some spatial queries, the 3D nature of the spatial scene can result in undesirable consequences when the problem is inherently two-dimensional. For example, an agent might want to determine how long it would take to move beneath an object that is in front of and above itself. In this case, simply extracting the distance will not work, since the vertical dimension must be ignored. In addition, if the agent is attempting to determine the closest obstacle to itself, it might be best to calculate the point-to-point distance from its centroid to the centroid of each obstacle, which can be done much more quickly than calculating the distance between polyhedrons. For these reasons, SVS supports spatial *object interpretations*. For most queries, the agent can specify that each object should be interpreted as a convex polyhedron (the default), a bounding box, a centroid, or two-dimensional versions of each of those types, where the projection of the object into its intrinsic xy-plane is used. Making shape interpretations available can increase the functionality of the system, and greatly speed it up when precise calculation over complex 3D objects is unnecessary.

4.5 Imagery

The information provided to Soar through perceptual item pointers and predicate extraction about objects that the agent can currently perceive is often not enough to allow general-purpose problem solving. Often, imagery must be employed. To use this capability, the symbolic system invokes a command which causes imagery processes to manipulate a concrete representation, and the results of the manipulation are symbolically inferred through predicate extraction. While imagery has been proposed in

[12] In the current implementation, this must be done through a multiple-step process (generating an image of the desired object with a new reference frame, and then extracting the relationship), but it should be eventually handled entirely in the predicate extraction system.

the past as a means for problem solving, the exact means by which images can be created in a task-independent manner have rarely been specified. One of the contributions of this work is exploring this problem. The current implementation of SVS inherits much of its functionality from a prior system, SRS. Two of the image creation processes in SVS, predicate projection and motion simulation, were previously explored in detail using that system (Wintermute & Laird, 2007, 2008).

Images, once created in the spatial scene or visual buffer, are thereafter treated identically to structures in those memories built by perception. In the discussion below, the term 'image' will refer to the structure being created, as in "the image is placed adjacent to the object", but it should be noted that once the image is placed, it becomes an object itself.

4.5.1 Predicate Projection

Creating a new spatial image often involves translating a qualitative representation of the image (present in symbolic working memory) to a quantitative representation in the scene. This problem has not been as well studied as predicate extraction (Chandrasekaran, 1997). We call the qualitative representation of a new image the *description* of that image. Our goal is to create a system with broad applicability, which requires a qualitative language for describing new images that is as expressive and general as possible. Broadly, there are two kinds of possible descriptions: direct and indirect.

An image created with an *indirect description* inherits its shape from an existing object, in the scene or in LTM. This shape is placed in the scene based on a set of abstract predicates describing the position of the object.

In SVS, the predicates currently are 'on', 'at', 'adjacent', and 'facing'. There is not a strong commitment to this particular set of predicates, but it is representative of the types of predicates than might be supported. Placing an image 'on' another object positions the image above and adjacent to that object, in the z-direction. This z-direction is currently that defined in the coordinate frame of the scene root, but the system could

be improved to allow the coordinate frame to be specified. Placing an image 'at' another object results in the image being centered at the center of that object. Placing an image 'adjacent' to another object is similar to 'on', but does not constrain the direction of adjacency. An image can be specified 'facing' another object. In that case, the intrinsic frame of reference of the image is aligned to point towards that object.

An indirectly described object is not guaranteed to have a valid position (an image cannot be both 'at' and 'on' the same object, for example). Underlying processing in SVS attempts to interpret the set of predicates, and adds the image to the scene if it can find a suitable position (or report an error otherwise). In many cases, indirect descriptions are underspecified, and the system arbitrarily chooses one of many images meeting the description. A previous implementation (Wintermute & Laird, 2007) included a much more comprehensive scheme for indirect imagery, but used a two-dimensional spatial representation. The underlying processing in that system solved complicated computational geometry problems in order to place the image, but this approach could not be easily extended to the three-dimensional case. Instead of extending SVS's indirect imagery power, alternative approaches are being explored to allow similar capabilities in the system through alternative means: the memory retrieval and motion simulation mechanisms (which will be covered shortly).

In addition to indirect descriptions, SVS also supports *direct* descriptions. In contrast to indirect descriptions, a direct description is always unambiguous – it describes only one possible image. An example is 'the image is the convex hull of objects **A** and **B**. For any diagram that has objects **A** and **B**, there is exactly one such shape. Other descriptions are those such as 'the image is the intersection of objects **A** and **B**'. For any objects **A** and **B** in the scene, the image may or may not exist, depending on their positions in the diagram, but if it does exist, there is only one. Thus, a direct description describes exactly one image, not a category of images.

The predicate projection processing described above is considered to be a fixed part of the architecture. There are currently no plans to allow new kinds of predicate projection

commands to be learned by the agent or provided as knowledge. However, there is not a strong commitment that the current library of available operations is appropriate or complete. It has served well for the tasks that have been addressed, but may change as the architecture evolves. From a theoretical point of view, the important aspect is that the system has this capability at all.

4.5.2 Memory Retrieval and Image Composition

Often, spatial images must be created based on objects in perceptual long-term memory. This involves the agent telling SVS to instantiate an object of a known type, at a known location. This is different than predicate projection—predicate projection works by the symbolic system *describing* the general required qualitative properties of the image, memory retrieval works by the symbolic system *referring to* specific items in long-term memory.

Using this capability, an agent can compose a new scene out of known parts (objects and transformations). In addition, memory retrieval can be easily combined with predicate projection to create images where the shape is based on a memory, but the location is qualitatively described. In Figure 9, the agent is generating an image based on this combination. The **`class-id`** of the imagined block is provided, which implicitly indicates a memory retrieval. A complete predicate projection command for the location of the new block is not shown, but the full command would indicate that the new block is on the table located adjacent to block **A**.

4.5.3 Motion Processing

While the previous approaches to image creation are powerful, it is difficult to see how they can encode the knowledge to solve spatial problems involving motion. For example, consider an agent that must predict the path of a bouncing ball. It is not obvious how the previously-discussed imagery components can achieve this. Anticipating the next location of a bouncing ball cannot be easily mapped onto retrieving a memory of a quantitative transformation, since that would require the agent to have previously observed a ball with similar horizontal and vertical velocities at

a similar place in its bounce, and the agent would have to somehow know to retrieve exactly that memory. Predicate projection will not work either, as it may be necessary to make a prediction that is more accurate than what can be described with the available qualitative predicates.

In SVS, this information can instead be encoded in the form of a continuous motion model. Motion models are stored in perceptual LTM, and are represented by the Motion Processing component in Figure 8. They can be applied to any object in the spatial scene, and are invoked with a step size. The model then creates an image of the object projected into the future for that amount of time, following its specific motion pattern. If the agent has a bouncing-ball motion model in its LTM, the model can track the ball and allow its motion to be projected into the future in imagery. Motion models have complete access to the contents of the spatial scene, and so can access the quantitative details necessary for precise simulation, and also have internal memory, allowing, for example, speeds of objects to be tracked based on observations over time. In general, motion models can transform the scene in arbitrary ways: there are currently no constraints on their implementation (adding more structure to the motion processing system is an area for future work).

Note that motion models have much in common with the concept of concrete controllers discussed in Chapter II (aspect A5), as they control simulation processes based on information that is in $\boldsymbol{R}_c$ (the spatial scene) but not $\boldsymbol{R}_a$ (Soar working memory). In the previous example, the agent reasons about the movement of an object that is not under its direct control. However, in SVS, the same basic motion processing system is used to control the agent's own low-level movement. That is, motion models can be concrete controllers in SVS. Those motion models that can be used as controllers are called motion controllers. Since motion controllers are motion models, it is necessary that they can be used to predict their own behavior—simulative imagery of concrete control (aspect A6) must be present for all motion controllers.

For example, a car motion controller can be used in SVS. When Soar uses the controller, it provides a perceptual pointer to the car object in the scene, and a pointer to a goal object, along with an indication of whether the motion is to be imagined or executed, and a time step in the imagined case. Based on the spatial scene, the controller can determine the body angle and position of the car. This information can be used to calculate a desired steering angle to set. To do this, the controller can determine the angle between the front of the car and the goal object, and steer in that direction, proportional to that difference, saturating at some maximum steering angle. When used in imagery mode, this angle, along with the time step, can be fed back in to a set of equations modeling the response of the car to the steering control, and the position and angle of the imagined car object can be determined. When used in execution mode, it can instead be output to the low-level action system. Even in execution mode, it may be useful to simulate the motion in parallel with execution, as this simulation can be used as part of a Kalman filter to assist in the control process (Grush, 2004). While this car controller is hypothetical, as the implemented system has not been used in real robots, the imagery aspects of it have been implemented and used (Wintermute, 2009a).

In many cases, the agent uses motion models not just for single-step prediction, but for longer simulations. Motion simulation can occur as a sequence of steps, executed by the symbolic level. Between steps, qualitative information can be queried from the scene and reasoned over. For example, an agent might need to determine how long it will take for a given obstacle to reach its current location. To do this, the movement could be stepped forward multiple times, with predicate extraction processes monitoring the scene for collisions between each step.

The motion processing system presents a broad framework for the representation of motion. With appropriate motion models, an agent can control and internally simulate its own movement. Using the same basic mechanisms, it can also simulate the movement of objects in the world, such as a moving obstacle in the example above, or a bouncing ball in another task. In certain circumstances, it might even use its own motion

controllers to simulate the movement of another agent. Motion simulation can even be used to solve problems that are not about motion in the world. Some of these applications are examined in detail in other work (Wintermute & Laird, 2008).

4.6 Aspects of the Theory in Soar/SVS

SVS, combined with the existing Soar architecture, forms an instantiation of the theory put forth in Chapter II. Here, the aspects of that theory are reviewed, and mapped to the implementation in Soar/SVS. As SVS was developed with a broader focus than this work, it also implements aspects that are not covered by the theory here, most prominently, visual processing and long-term memory. Lathrop (2008) argued for the utility of visual processing, but an analysis of the utility of perceptual long-term memory (especially in contrast with Soar's symbolic memories) must be left to future work.

A1. Bimodality

- Two representations of information derived from perception are present, $\boldsymbol{R}_a$ and $\boldsymbol{R}_c$.
- Representation $\boldsymbol{R}_c$ contains more perceptual information than $\boldsymbol{R}_a$—it makes more distinctions between states of the world. If $\boldsymbol{R}_c$ encodes spatial locations of objects in the world, it is a *concrete spatial* representation.
- Processes can encode information in $\boldsymbol{R}_a$ based on $\boldsymbol{R}_c$ (through *high-level perception* processes).

In Soar/SVS, Soar's working memory encodes an abstract representation, and the spatial scene encodes a concrete spatial representation (the visual buffer encodes a separate concrete representation, but it is outside the scope of this work). High-level perception in SVS is implemented by the perceptual pointers that identify objects in the scene and the predicate extraction processes.

A2. Concrete routines

- Processes can cause changes to representation $\boldsymbol{R}_c$ based on its existing contents (they can locally manipulate it).

All of the imagery mechanisms described above manipulate the scene in this way.

A3. Imagery

- Concrete routines can be invoked by processing in $\boldsymbol{R}_a$, and result in persistent changes to $\boldsymbol{R}_c$. These are imagery processes.
- Via high-level perception, results of imagery will be reflected in $\boldsymbol{R}_a$.

The imagery mechanisms in Soar/SVS all satisfy this description. It should be noted that *all* concrete routines in the system are imagery processes, none are initiated outside of Soar or cause transient results.

A4. Simulative imagery

- Some imagery operations simulate future states of the world in terms of $\boldsymbol{R}_c$.

The imagery mechanisms in Soar/SVS are expressive enough that simulation is possible, as will be demonstrated in later chapters.

A5. Concrete controllers

- External actions can be contingent on information in $\boldsymbol{R}_c$ but not in $\boldsymbol{R}_a$. Modules that generate these actions are called concrete controllers.

Motion controllers in SVS are proposed to meet this requirement, although external actions of this sort remain unimplemented.

A6. Simulative imagery of concrete control

- Some simulative imagery operations simulate the effects of concrete controllers.

All motion controllers in SVS must have simulation capability.

A7. Architectural representation conversion

- High-level perception and imagery are supported by fixed (although possibly parameterized) mechanisms.

Fixed mechanisms underlie the high-level perception system—no knowledge is involved in predicate extraction, other than to parameterize the process (determine the queries). Hypothetically, knowledge is involved in object recognition, as the agent must be able to learn to recognize new objects. However, if recognition were to be implemented, the process would likely be supported by architectural mechanisms. Similarly, in the action system, predicate projection is a fixed process where the only role of knowledge is to parameterize the commands in Soar, while

memory retrieval and motion simulation involve access to more forms of knowledge, but are still supported by architectural mechanisms.

A8. Perception/action reuse

- Some types of perceptual information arrive in $\boldsymbol{R}_a$ only via $\boldsymbol{R}_c$.
- Common high-level perception mechanisms operate over structures created by imagery and low-level perception in $\boldsymbol{R}_c$.
- Some imagery processes share mechanisms with those used to generate external actions.

In Soar/SVS, spatial information does not enter Soar without being processed by SVS. All high-level perception processes apply to both imagined and perceived objects, and motion controllers that execute external actions can be re-used internally for imagery.

Chapter V - Reinforcement Learning Agents in Soar/SVS

To aid in evaluating the architecture and its underlying theory, implemented agents are necessary. In order to reduce the factor of hand-programmed knowledge in this evaluation, an integration between imagery and reinforcement learning in Soar/SVS is described in this chapter, so that all control knowledge is learned as opposed to programmed. The aims of this discussion are threefold: to provide further explanation and evaluation of the architecture through demonstrations of implemented agents, to provide evidence of the benefits outlined in Chapter III, and to connect this work with concepts and related work in the area of reinforcement learning.

5.1 State Abstraction and Imagery in Reinforcement Learning

Work in reinforcement learning typically models the task being addressed as a Markov Decision Process (MDP). An MDP consists of a set of states, a set of actions, a function encoding transition probabilities from one state to another (given an action), and a function encoding the expected immediate reward for each transition. $\mathcal{P}_{ss}^{a}$, indicates the probability of transitioning from a state s to another state s' with action a, and $\mathcal{R}_{s}^{a}$ indicates the expected immediate reward for action a in state s.[13] Here, we will assume that an agent is actively engaged in the problem, and has no initial knowledge of the transition probabilities or reward distribution. At every time step, the agent observes a state s_t, and selects an action a_t. The environment then transitions and provides the agent with a reward r_{t+1} at the next time step.

Essentially, an MDP describes a large state space, where actions probabilistically cause an agent to move between states and receive rewards. Each transition in an MDP must be conditionally independent of previous transitions, given the state the agent is

[13] This notation follows Sutton & Barto (1998). The notation $\mathcal{R}_{ss}^{a}$, for immediate reward is used more often in that book, but the two forms are equivalent: $\mathcal{R}_{s}^{a} = \sum_{s'} \mathcal{P}_{ss'}^{a} \mathcal{R}_{ss'}^{a}$, (p. 84).

transitioning from and the action; this is the Markov property. A reinforcement learning agent learns a policy (a mapping of states to action choices) to maximize its expected future reward. Often, a discount rate will be used, causing the agent to value earlier rewards more than later rewards, and the agent will instead learn to maximize expected future discounted reward. More detail about reinforcement learning can be found elsewhere (e.g., Sutton & Barto, 1998).

The MDP formalism can provide an objective measurement of what it means to have a "good" state representation for a task: a good state representation makes the transition probabilities Markovian (they have the Markov property), and allows for policies to be represented with an expected future reward that is as large as possible, meaning that it captures all details of the world necessary to select the best action. However, it is also important that this state representation be compact: learning can quickly become intractable if the state space is large.

These points can be seen in simple domains like the blocks world, as was touched upon in Section 2.2. If the agent encodes the complete spatial state of the blocks (their bounding coordinates in continuous numbers), the representation is Markovian and allows for optimal policies to be encoded. However, if the agent is solving multiple instances of blocks world problems where the block dimensions very minutely between instances, encoding the complete spatial state results in a situation such that the agent rarely experiences repeated states. Repeated experience is necessary for learning, so this agent would perform very poorly. Compact state representations lead to repeated experience in terms of those states, and hence faster learning.

To make a more compact learning problem, allowing faster learning, state aggregation can be used[14]. Formal techniques exist for determining equivalent states in an MDP, and the size of a given MDP can be reduced by checking for these equivalencies and

[14] Function approximation is another technique used in reinforcement learning to deal with large continuous state spaces. Given the symbolic assumption discussed in Section 2.1, it will not be considered here, however, the subject will be discussed in Section 7.4.

aggregating equivalent states into abstract states (Ravindran & Barto, 2002; Givan et al., 2003; Li et al., 2006). Alternatively, it is possible to take an architectural view of the issue, and define a perception system that implicitly aggregates states together as it builds an internal abstract representation. In the case of blocks world, the perception system can build predicates such as **on(A,B)** that form a state representation that is Markovian, allows for maximum reward to be achieved, and is minimal.

If state abstraction results from perception in this way, in order to create an agent able to induce compact MDP representations in arbitrary problems, the general perceptual abstraction problem must be solved. As was discussed in Chapter III, the imagery architecture proposed here provides benefits that help mitigate the perceptual abstraction problem. Specifically, the architecture provides mechanisms that allow an agent to encode abstract properties that capture movement and nonlocal interaction (B1), and to allow task-specific abstract properties to be encoded by a task-independent perception system (B2).

5.1.1 The Pedestal Blocks World

To provide a concrete example of these benefits in a reinforcement learning setting, the pedestal blocks world task (Figure 12) outlined in Section 2.2 will be used. Here, the agent is presented with a table and three blocks. There are six pedestals fixed to the table upon which the blocks can be placed. The goal is to place the blocks on the pedestals in the correct order (**A** to the left, then **B**, then **C** to the right). The agent moves each block to a pedestal, first **A** then **B** then **C**, and then receives some reward.

Recall that a reward of 100 is received for placing the blocks in the correct order, but 10

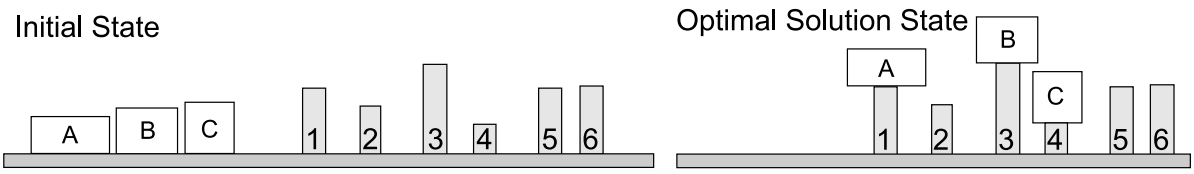

Figure 12: A Pedestal Blocks World task instance and its optimal solution.

points are deducted from the reward for each empty platform to the left of **C**. If the agent places a block where it cannot fit, it receives a reward of -100 and the task ends. If the agent places the blocks on the pedestals without a collision, but the ordering is wrong, a -10 reward is received.

An optimal policy for solving the problem is apparent: place block **A** on the leftmost pedestal where it will fit, place **B** on the leftmost pedestal right of **A** where it will fit, and place **C** similarly. However, an agent solves many instances of this task. In each instance, the positions and heights of the pedestals, along with the dimensions of the blocks, differ. Because of this, the actual moves needed to optimally solve the problem differ from instance to instance. Assume that the agent views the task on a computer screen, and interacts by pressing buttons to indicate the pedestal where each block should be placed. The display updates after each block is moved. This problem then has a simple formalization: given pixels, button choices must be output.

5.1.2 Perceptual Abstraction in Pedestal Blocks World

Taken at its basic definition, the problem is an MDP where each set of pixels constitutes an individual state. Of course, state aggregation would be very valuable here, since otherwise too many states would be present for learning to be tractable.

As with the pegged blocks world task in Chapter II, a perception system providing a standard blocks world encoding of the state in terms of abstract predicates like **`on(A,table)`** or **`on(B,pedestal1)`** is inadequate, since collisions cannot be predicted, and the best policy would have a low expected future reward. The nonlocal interaction of the block with the surrounding objects at its new location is critical to capture in order to induce a correct state aggregation.

Soar/SVS can use its task-independent high-level perception system to encode the standard blocks world **on** predicates by composing them out of the primitive available in the predicate extraction system. Details of this composition can be found elsewhere (Wintermute, 2009b). Predicate projection in the system can also be used to imagine the blocks at their new locations, and high-level perception can be applied to the

imagined scene to determine whether actions result in collisions. These collision predictions are task-specific properties of the current state. Through these means, the Soar/SVS agent can infer task-specific symbolic information capturing non-local interaction using its task-independent high-level perception system, demonstrating benefits B1 and B2.

An agent has been built to use this abstract state information with reinforcement learning in this task. The abstract state consists of **on** predicates describing the current scene, along with predicates encoding whether or not each action will result in a collision (e.g., **`collision_if_moved(B,pedestal2)`**). As imagery is used to add information to the abstract state, this agent will be called an imagery-augmented state abstraction agent.

State abstraction here is used in conjunction with a table-based Q-learning algorithm to learn a policy. For each state-action pair that a table-based Q-learning agent encounters, it learns the expected discounted future reward for taking that action and following the optimal policy—this is called the Q value of the action. Soar's existing reinforcement learning system implements the learning algorithm (Nason & Laird, 2005).

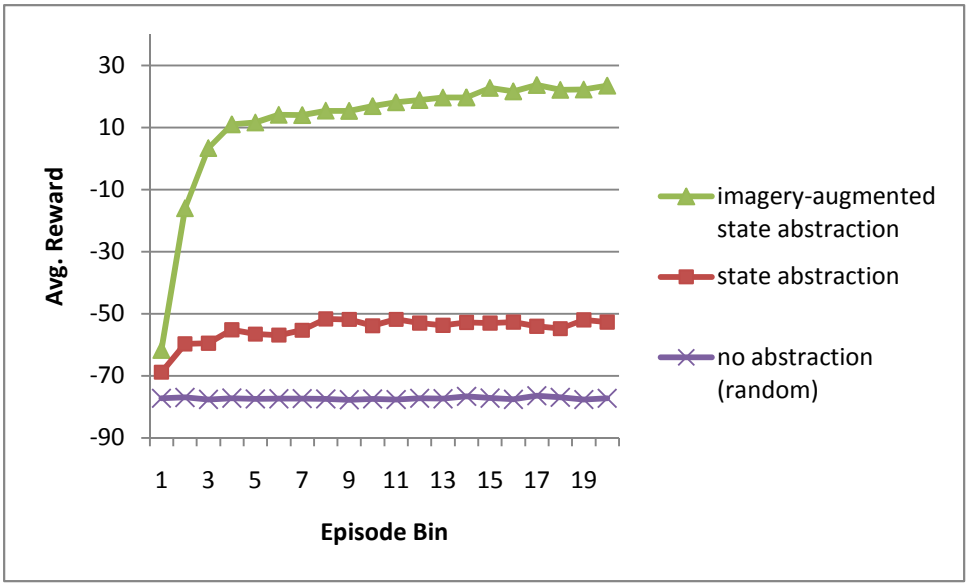

Figure 13: Results of learning in pedestal blocks world showing advantage of imagery-augmented state abstraction.

To verify that the overall system works as described, experiments were run to compare the performance of an agent using imagery-augmented state abstraction versus an agent that can only encode the **on** predicates describing the current scene (non-imagery state abstraction) and an agent that takes random actions, mimicking one that learns in terms of the raw (unabstracted) pixel states[15]. Figure 13 shows the results of this experiment. 25 trials were run of 10,000 episodes each. The sizes of the blocks and positions of the pedestals were randomized for each episode, each was a spatially-unique instance (both the imagery and non-imagery conditions used the same instances). Epsilon-greedy exploration was used, with parameters of $\alpha = 0.3, \varepsilon = 0.1$, and $\gamma = 0.9$. Total reward per episode was collected, bins of 500 adjacent episodes were grouped together, and reward was averaged across all trials and all episodes in the bin and across all trials

As can be seen in the figure, in this case, learning using task-specific abstract information derived from imagery results in better performance, both in terms of learning speed and the quality of the final policy, when compared to similar states

[15] Since the task has no state repetition within an instance, and at the pixel level, each instance tested has a unique set of states, no learning would ever occur in a non-aggregating agent, resulting in random performance.

abstracted without using imagery augmentation. Both of these approaches outperform learning directly in terms of concrete (pixel-based) states. Since imagery allows the agent to encode useful task-specific abstract properties that capture non-local interaction, these data demonstrate benefits B1 and B2 and provide evidence that the relevant aspects of the architecture are working to mitigate the perceptual abstraction problem.

5.2 State Abstraction, Action Modeling, and Imagery in Reinforcement Learning

The previous section demonstrated the use of the architecture in aiding reinforcement learning by allowing a more compact MDP representation of the problem to be induced. A benefit of the architecture that the previous demonstration does not cover is that simulative imagery allows for abstract action modeling (B3). As discussed in Section 3.3, the sort of action modeling imagery allows may be difficult to integrate with existing model-based RL techniques, as imagery predictions are often nondeterministic in terms of abstract states. However, the action modeling capability simulative imagery allows can be integrated with reinforcement learning in other ways.

Motivation for this integration can come from the previous example in pedestal blocks world. The imagery agent in this task infers an abstract state by imagining the potential actions, and determining if collisions would occur, encoding predicates like **`collision_if_moved(B,pedestal2)`**. These predicates were used as a means of describing the current state. However, they can also be interpreted as predictions about future states—the results of an action model. The action modeling knowledge implicit in these predicates is not leveraged by the agent in that section, however.

The insight for the approach that will be taken here is that, with the above state representation, the agent is conditioning the value of each particular action on the predictions for all actions. However, a predicate like **`collision_if_moved(B,pedestal2)`** is much more relevant for determining the value of moving **B** to **`pedestal2`** than it is for determining the value of another action,

like moving **B** to **pedestal5**. In addition, if the agent has knowledge about the state that will result from an action, that information can be much more relevant than information about the current state when considering the value of an action.

Based on these insights, a new technique for integrating reinforcement learning with imagery was developed, ReLAI (Reinforcement Learning with Abstraction and Imagery; Wintermute, 2010). In a ReLAI agent, the value of an action is determined solely by the next abstract state predicted to result from that action. Technically speaking, ReLAI involves an aggregation of state-action pairs, rather than an aggregation of states. That is, individual entries in the table of values learned by Q-learning are aggregated, rather than states of the MDP. The aggregate (or *category*) that a state-action pair belongs to is determined by the predicted next abstract state that will result from it.

To prevent confusion, the standard state abstraction approach used above, where Q-learning occurs as normal but within an abstract state space, will be called *direct* state abstraction. Direct state abstraction agents may or may not use imagery augmentation to construct the state. State abstraction is used within ReLAI agents, but interacts differently with the learning algorithm.

To see the difference between imagery-augmented direct state abstraction and ReLAI, consider the following circumstance: block **A** is on **pedestal1**, and blocks **B** and **C** are on the table, so **B** will be moved next. The agent predicts that moving **B** to **pedestal2**, **pedestal3**, or **pedestal6** will not cause a collision, but moving to **pedestal4** or **pedestal5** will. The best action here is to move **B** to **pedestal2**. To find the learned value of that action, the imagery-augmented direct state abstraction agent in the previous section would add the imagery predictions to its current state, and look up an entry in its table using the complete state-action pair:

```
state=[on(A,pedestal1) on(B,table) on(C,table)
 no_collision_if_moved(B,pedestal2)
 no_collision_if_moved(B,pedestal3)
 collision_if_moved(B,pedestal4)
 collision_if_moved(B,pedestal5)
 no_collision_if_moved(B,pedestal6)]
action=[move(B,pedestal2)]
```

A ReLAI agent, on the other hand, would look up a learned value based only on the predicted next abstract state for the action, or:

```
[on(A,pedestal1) on(B,pedestal2) on(C,table) collision(false)]
```

As is apparent, the ReLAI agent takes into account less information when looking up (and learning) Q values: it has a more compact learning problem. If the right information is captured by the predictions ReLAI uses (as will be discussed), the algorithm can learn the optimal policy faster than the direct state abstraction agent. The ReLAI algorithm as instantiated in Soar/SVS is shown in Figure 14.

```
for each episode
  for each step in the episode
    perceive the concrete state s and any reward,
    store s in the spatial scene
    for each action a
      use imagery to simulate a in the spatial scene
      apply high-level perception to the imagined
      scene, derive the next abstract state A(s')
      lookup the learned value of a in s based on
      the category of (s,a), which is A(s)
    given the current action values and the
    reward, apply a Q-learning update to the
    category of the previous action (if any)
    choose an action using epsilon-greedy policy
  repeat until s is terminal
repeat for all episodes
```

Figure 14: The ReLAI algorithm as instantiated in Soar/SVS.

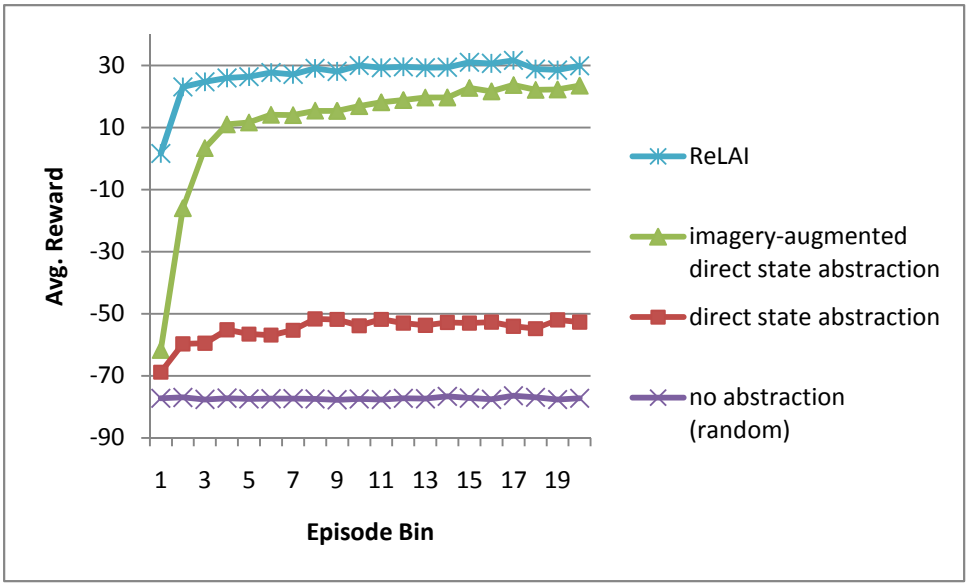

Figure 15: Results of learning in pedestal blocks world showing advantage of ReLAI.

Figure 15 demonstrates the performance of a ReLAI agent in pedestal blocks world compared to the agents introduced in the previous section. Experimental details are the same as in the previous section. These data demonstrate that ReLAI can learn much faster than using the same prediction information as part of a state representation and using direct state abstraction. This is because ReLAI is able to leverage the benefit of simulative imagery to model actions (B3). Since the agent knows that prediction information is information about a particular action in the current state, rather than just a generic property of the current state, the size of the learning problem can be greatly reduced, resulting in faster learning.

5.2.1 Correctness in ReLAI

While the previous example provides empirical evidence that ReLAI allows an agent to learn good policies, theoretical analysis can reveal general principles about the technique that can move the evaluation of the technique (and the architecture that supports it) beyond what is possible with demonstrations alone.

To better understand ReLAI, an analysis has been carried out to show under what conditions Q-learning using ReLAI will be guaranteed to converge to the optimal policy.

This analysis will be outlined here, but more detail is provided in a separate paper (Wintermute, 2010).

ReLAI is a special case of Q-learning using a state-action aggregation. In a standard Q-learning agent, assuming the problem is an MDP, given enough experience and an appropriate exploration policy, the agent will learn the optimal value function $Q^*(s,a)$, which is defined as $\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \, max_{a'} \, Q^*(s',a')$. Previous work has shown that Q-learning with state-action aggregation converges to the optimal Q^* function under the same conditions when all (s,a) pairs in the same category have the same Q^* value (Goetschalckx, 2009).

Properties of a function that assigns categories to state-action pairs will be examined here. This function, $C(s,a)$, returns a symbol representing the category of (s,a). An appropriate function only assigns two (s,a) pairs to the same category if their respective Q^* values are the same, and will hence allow Q-learning to converge. Sufficient conditions such that a function C is appropriate have been shown elsewhere (Wintermute, 2010). These conditions are that, for all states and actions, the reward received for a transition is independent of (s,a), given $C(s,a)$, and that categories of actions in a given state are independent of the (s,a) pair that led to that state, given the category of that pair. That is, Q-learning with state-action aggregation will converge to the correct policy if these equations are always true:

$$Pr\{r_{t+1} = r | s_t, a_t, C(s_t,a_t)\} = Pr\{r_{t+1} = r | C(s_t,a_t)\} \quad (1)$$

$$Pr\{C(s_{t+1},a) = x | s_t, a_t, C(s_t,a_t)\} = Pr\{C(s_{t+1},a) = x | C(s_t,a_t)\} \quad (2)$$

Now, the special case of ReLAI can be considered: where the category of an (s,a) pair is determined by the predicted next abstract state resulting from action a in concrete state s. How, then, should these abstract states relate to the underlying concrete states? Since Equations 1 and 2 are sufficient for appropriate (s,a) aggregation, and thereby allow Q-learning to converge, we re-examine them in the situation where the category of (s,a) is a correct prediction of the abstract state that will follow from it.

An abstraction function $A(s)$ returns the abstract state corresponding to concrete state s. In the examples here, this function is high-level perception applied to the contents of the spatial scene in SVS—it is the process that converts the concrete representation to an abstract representation.

Assume that predictions are correct: in all cases $C(s_t, a)$ is what $A(s_{t+1})$ would equal if action a were to be taken.[16] Under this assumption, Equation 1 holds if and only if the reward received for a transition is always independent of (s, a), given the next abstract state:

$$Pr\{r_{t+1} = r|s_t, a_t, A(s_{t+1})\} = Pr\{r_{t+1} = r|A(s_{t+1})\} \quad (3)$$

Similarly, under this assumption Equation 2 holds if and only if the abstract state that would result from taking some action a in s_t (which is $C(s_t, a)$) is independent of the previous state and action (s_{t-1}, a_{t-1}), given $A(s_t)$. This implies that the next abstract state must be independent of the previous (s, a) pair, given the current abstract state and action:

$$Pr\{A(s_{t+1}) = x|s_{t-1}, a_{t-1}, A(s_t), a_t\} = Pr\{A(s_{t+1}) = x|A(s_t), a_t\} \quad (4)$$

Equations 3 and 4, along with prediction correctness, can then be regarded as sufficient conditions for convergence with ReLAI[17]. This means that ReLAI can use abstraction functions where $A(s_{t+1})$ is not independent of s_t given $A(s_t)$, but is independent of s_{t-1}. This stands in contrast to direct state abstraction techniques, where $A(s_{t+1})$ must typically be independent of s_t given $A(s_t)$ for guaranteed convergence (e.g., Ravindran & Barto, 2002; Givan et al., 2003). The ability to use state abstractions where $A(s')$ is

[16] This means that the agent correctly predicts all actions it takes, and $C(s_t, a_t) = A(s_{t+1})$ at all times, but also that the agent correctly predicts actions it does not actually take.

[17] Note, however, that Equation 4 does not strictly imply Equation 2. Equation 4 only covers actions the agent actually takes, not all *possible* actions as Equation 2 requires. There might be some abstraction function that, when used with a particular policy, meets Equation 4 for the actions taken, but would not have for other actions. However, that possibility will not be considered here.

not independent of s given $A(s)$ allows for less constraint on the high-level perception system used to induce the abstraction function.

Even so, these assumptions can be difficult to match. In the example pedestal blocks world problem as presented above, equation 3 is met, since the reward for a transition is completely determined by the resulting abstract state. However, equation 4 is not met: since the problem is deterministic, all information necessary to exactly predict future states is implicit in the initial state, but is not captured by the abstraction. Future abstract states are then never independent of *any* previous concrete state. A simple manipulation of the domain, however, reveals that ReLAI will still work in this task, as the data indicate.

Consider an alternate version of the domain, where after each block is placed, the agent is transported to a random instance of the task sharing the same abstract state (**on** and **collision** predicates). That is, after each action, the spatial details of the problems are randomly changed without changing the abstract state. In this alternate domain, the reward for a transition is still determined by the resulting abstract state, so equation 3 still holds. In contrast to the original domain, though, the next abstract state resulting from a transition here is independent of the previous concrete state, given the current abstract state, so equation 4 is met.

In this alternate version of the task, the optimal policy is the same: greedily place the blocks as far to the left as possible without collisions. In addition, viewed in terms of the inputs to the learning algorithm (rewards and abstract states), the experience of the agent in the actual domain is virtually identical to what it would experience in the alternate domain[18]. Since the agent would learn the optimal policy in the alternate

[18] The exception is that, in the real domain, there is some correlation between potential collisions for one block and for another, since pedestal dimensions effect both calculations. For example, assume the agent infers that moving **blockA** to **pedestal2** causes a collision, and moves it to **pedestal1** instead. The agent next considers **blockB**. Since the **blockA**/**pedestal2** collision is known, it is now more likely that moving **blockB** to **pedestal2** will also cause a collision, as it could be that the pedestals are very close together. In the alternate problem, since pedestal dimensions change after each move, this correlation is not present. It will be assumed that this minor difference does not substantially affect convergence.

domain, the optimal policies are the same, and the agent's experiences are consistent with the alternate domain, the optimal policy can be learned in the actual domain.

5.2.2 ReLAI and Perceptual Abstraction

In addition to benefiting from action modeling (B3), ReLAI agents retain the perceptual abstraction-related benefits (B1 and B2) demonstrated by the imagery-augmented direct state abstraction agent in Section 5.1. Beyond what is demonstrated in that section, though, the theoretical analysis here can add to the understanding of the benefit of task-specific abstract property generation (B2).

As demonstrated by both the direct state abstraction and ReLAI agents, using imagery in pedestal blocks world can allow the task-independent high-level perceptual system in SVS to infer task-specific properties (collisions in future states), resulting in better performance, and demonstrating benefit B2.

Generalizing this result beyond that particular task, if state abstraction is supported by high-level perception, an agent architecture might have some fixed library of perceptual processes, for example, SVS's predicate extraction system. Since these processes can be used in any task, they are task-independent. This library won't work well in all tasks when used with direct state abstraction, assuming the poverty conjecture is true. However, when used with simulative imagery, that same library can provide further useful properties. Since these properties are calculated via simulations of the actions specific to that particular task, they can be considered task-specific properties.

In this scheme, by encoding different properties into an abstract state, an agent induces an abstraction function $A(s)$. To solve a particular task, an agent's architecture must support creating an abstraction function for that task. Both imagery augmentation and ReLAI share a common benefit of allowing an agent with a fixed library of perceptual processes to address more problems than would otherwise be possible; however, further discussion will focus on ReLAI, since it also incorporates action modeling benefits (B3), and has been theoretically analyzed in more detail.

The theoretical results for ReLAI reveal that it can increase the usefulness of a given set of abstraction functions. Compared with what is needed for direct state abstraction, equation 4 shows that ReLAI allows an agent to use abstract states that relate in a fundamentally different way to the concrete states of the problem, with guaranteed convergence of learning to the optimal policy. Because of this, abstraction functions that do not meet the requirements for correct direct state abstraction in a given problem may meet the requirements for correct ReLAI state abstraction.

For instance, Li et al. (2006) recently presented a comprehensive theory of methods for direct state abstraction, describing five abstraction classes of increasing generality, and grouping abstraction techniques into those classes. Of those classes, the most general for which Q-learning convergence is guaranteed is called Q^*-*irrelevant*. Here, the only requirement is that all concrete states in the same abstract state have the same Q^* value for all actions. However, ReLAI allows convergence with abstraction functions that are not Q^*-irrelevant. For example, the abstraction function used in pedestal blocks world is in not Q^*-irrelevant. All initial states of the problem are grouped together, regardless of whether moving **A** to **`pedestal1`** will or will not cause a collision, situations that clearly effect the Q* value of the action **`move(A,pedestal1)`**. This is an example of how the different relationship between concrete and abstract state spaces with ReLAI compared to direct abstraction allows different abstraction functions to be successfully used.

While theoretically interesting, taken at face value, the formal requirements for ReLAI do not appear to be very practical. Even for the simple pedestal blocks world task, as examined above, the requirements are not strictly met[19]. However, rather than treating these requirements as an objective to meet, they may have more practical value as an ideal to approximate. While exactly satisfying the equations guarantees convergence to the optimal policy, a reasonable hypothesis is that, to the degree the equations are

[19] A simple task where the requirements for ReLAI are more straightforwardly met is presented elsewhere (Wintermute 2010). This task also uses a non-Q^*-irrelevant abstraction function, supporting the argument in the previous paragraph.

approximated, performance will approach the optimal ideal. Further theoretical work may produce formal measures of approximation, but, as will be demonstrated, use of the equations as an informal guide to constructing state representations can lead to empirical gains. Roughly, a good state abstraction for use with ReLAI should capture as many of the details possible which determine immediate rewards leading into a state (for equation 3), but need not capture all information necessary to choose an action, as long as a one-step lookahead in abstract state space provides the necessary information (as equation 4 allows, since the consequences of actions can be dependent on details in the concrete state but missing from the abstract state).

From these reasons, then, a given set of abstraction functions can be more useful with ReLAI than with direct state abstraction. Abstraction functions that do not meet the formal requirements for correct direct state abstraction in a given task may meet the requirements for ReLAI, and empirically, abstraction functions that do not work well with direct state abstraction may work well with ReLAI. The architectural structures necessary for direct state abstraction are a subset of those necessary for ReLAI, so any agent capable of ReLAI is also capable of direct abstraction. This means that ReLAI increases the breadth of tasks an agent will be able to address with a task-independent perception system. The reason ReLAI has this advantage over direct abstraction is that it captures additional task-specific abstract properties through imagery (even though these properties are not explicitly used in a state representation, as they are with imagery-augmented direct abstraction). Overall, this amounts to strong support for benefit B2, that imagery can mitigate the perceptual abstraction problem by allowing task-specific abstract properties to be encoded by a fixed perception system, increasing the generality of the architecture.

There is some cost to using ReLAI compared to direct abstraction, since low-level imagery knowledge is necessary to simulate actions, and since imagery processing takes time. However, in many tasks, the benefit to be gained in terms of achieving better performance with a fixed perception system clearly outweighs these costs.

5.3 ReLAI in Complex Problems

In this section, ReLAI in Soar/SVS is applied to complex arcade game tasks. In addition to providing further demonstrations of applications of ReLAI, these examples also demonstrate the comprehensive capabilities of the implemented architecture.

Inspired by other work using arcade games as a source of AI problems (e.g. Agre & Chapman, 1987; Diuk et al., 2008), three tasks are demonstrated using games for the Atari 2600 system. The original Atari games are used (run in an emulator) – they have not been reimplemented.

For each task, an abstract state representation computable by the high-level perception system in SVS has been chosen. A ReLAI agent and a direct state abstraction agent have been created for each task using the given representation, and their performance is compared. Each of these comparisons show an advantage for ReLAI, demonstrating that a given state abstraction can provide better performance with ReLAI than with direct state abstraction, even if it does not meet the formal convergence requirements of either technique. In addition, all ReLAI agents perform much better than random, providing simple demonstrations that imagery in Soar/SVS can be productively used to address these tasks.

These tasks work well as demonstration domains for several reasons. First, each is much more complex than a blocks world task: each involves many objects, which are in constant motion, and which interact with the reward function in different ways (e.g., a player icon to be controlled, a prize to collect, or an enemy to avoid). Second, each task has different interesting spatial interactions from the others, so the set captures some of the diversity of spatial tasks, posing a challenge to address all of them using a common architecture. Finally, the tasks are nonarbitrary, they were invented by others who were not concerned with demonstrating the claims made here.

Additionally, viewed as demonstrations of the architecture in general, the use of reinforcement learning in these agents eliminates the parameter of policy knowledge from those demonstrations.

All games are run in an Atari 2600 emulator. A low-level perception system has been constructed which segments, identifies, and tracks relevant objects based on the pixels output by the emulator. The recognized objects are input to SVS, where they are added to the spatial scene. The perception system is not completely general-purpose: human tuning is needed to provide game-specific parameters (including object labels), and some game-specific perceptual code is needed to augment what is provided by the generic interface. Outside of this low-level perceptual interface; however, the architecture is unchanged from what is presented in Chapter IV.

The action interface is customizable, but here, all agents are allowed to choose an action once every 15 game frames (four per second). This value was chosen as a (very) rough estimate of human reaction time. The experiments here examine the quality of learning that the agents achieve (and not reaction speed), so the emulator is paused while the agent processes the perceptions and chooses an action.

Following the algorithm in Figure 14, each ReLAI agent simulates all of its (game-specific) action choices, and applies the (game-specific) abstraction function to each resulting imagined state, which is then used in the learning algorithm as the category of the action. In many cases, though, the imagery process can be decomposed to increase efficiency. The changes to the state of the game at each step can be divided into two categories: those changes that are caused by the agent, and those that occur independently of the action choice (*environmental* changes). Rather than inferring the imagery state for each action independently, an agent can first simulate the environmental changes, and then successively overlay that simulation with simulations of each of the agent's own actions.

To perform imagery, all of these agents rely heavily (but not exclusively) on simulating linear motion of objects. A linear translation motion model in SVS supports this, which

Figure 16: Perceptual information in the game Frogger II, including object labels.

can be instantiated to track and project forward the motion of each relevant object[20]. By tracking the movement of a single object, the instantiated motion model learns its velocity, knowledge that it uses when a simulation of future movement is requested.

5.3.1 Frogger II Agent

Figure 16 shows the perceptual information provided by the emulator for the first game that will be addressed, Frogger II[21] (Parker Bros., 1984). In the figure, the perceptions are overlaid with object outlines and category names are provided by the generic low-level perception system.

The agent has a simple goal of navigating the frog (bottom center of the figure) to the area below the raft objects at the top of the screen, without colliding with any of the

[20] Each such instance of the motion model is independent of the others. This is similar to a situation where an agent imagines two instances of the same long-term memory object. While the objects are based on the same LTM prototype, they are independent as instantiated in the spatial scene.

[21] All images are copyright of their respective owners.

moving obstacles or leaving the play area. This is a simplification of the complete game, which would involve solving multiple screens, playing through multiple lives, collecting bonuses, etc. Without considering the rest of the game, though, this task is still very difficult. The frog has five actions: move in four directions, or do nothing. There is a slow current in the water pulling the frog to the right, so inaction still results in motion.

The position of the frog is discrete in the vertical direction (there are 9 rows to move through), but many horizontal positions are possible due to the current. Most of the obstacles move continuously at uniform speed to the right or the left, although some move vertically or diagonally. Obstacles are constantly appearing and disappearing at the edges of the screen. This is an episodic task, and the initial state of the game differs across episodes (the obstacles start in different positions), so memorization of an action sequence will not work. Rather, a general policy must be learned.

A reward function similar to that of the game score has been implemented: there is a reward of 1000 for winning (reaching the top row), and -1000 for losing (colliding with an obstacle or leaving the area). There is a reward of 10 for moving up, and -10 for moving down. At every time step, there is also a reward of -1 to encourage short solutions.

To apply ReLAI in this task, imagery must be capable of simulating future states of the game. Motion models in SVS support this capability. All of the objects in the game can be assumed to be moving linearly at a constant velocity, and, as mentioned above, a motion model has been implemented to track and project forward such movement. For the movement of the frog itself, the agent has been provided with background knowledge in the motion model about how the frog's controls change its position (for example, that an "up" action moves it 12 units in the +y direction).

The abstract perceptions used by ReLAI in this task encode the following information in working memory:

- a predicate[22] encoding the vertical position of the frog: one of the 9 rows that define the legal play area

- a predicate encoding the horizontal position of the frog: a left, middle or right region

- a predicate encoding whether or not the frog currently intersects an obstacle

- a predicate encoding whether or not an obstacle (or screen edge) is adjacent to the frog in each of the four directions.

As implemented, horizontal and vertical discretizations are achieved by augmenting the perceptual information in Figure 16 with objects outlining the relevant regions, and using predicate extraction to determine what regions the frog intersects. Collisions are simply detected through predicate extraction. Directional obstacle adjacency is determined by first using predicate extraction to determine which obstacles are located in the appropriate direction of the frog, and then extracting the distance from the frog to any matching obstacles. If the distance is less than a threshold (10 pixels, about the same as the inter-row distance), the obstacle is deemed adjacent in that direction.

As a state representation, this abstraction loses potentially useful information, and is not Markovian (since the agent could make better decisions by remembering where it has seen obstacles in the past). However, it is compact, and just as important, it can be composed from the simple perceptual operations available in the architecture.

The same perceptual abstraction function is used in both a direct state abstraction agent and a ReLAI agent. At each step, the ReLAI agent uses imagery to project forward the motion of the obstacles near the frog, along with the effect of each action on the frog. The abstract state information above is then inferred for each imagined state. In addition to abstract perceptions, in this task the ReLAI agent also encodes the proposed action as part of the abstract state. This is because perceptions about the next state alone cannot capture the immediate reward for the transition, as Equation 3 requires,

[22] As in the rest of this thesis, the term “predicate” here is simply shorthand for “symbolic structure”.

since moving up or down a row effects reward (not just being in a particular row). However, the last action taken is not useful as part of the other agent's state, so it is not included there.

For ReLAI, the requirement that the abstraction captures immediate reward (Equation 3) is met, and the requirement that predictions are accurate comes close to being met, only missing a few cases where moving objects do not follow a constant velocity or disappear unexpectedly. The requirement on state independence (Equation 4) is not met: $A(s_{t+1})$ is not strictly independent of s_{t-1}, given $A(s_t)$, so convergence to Q^* isn't guaranteed. However, unlike state aggregation, ReLAI is robust to abstractions where $A(s_{t+1})$ *is* dependent on s_t given $A(s_t)$, which can be beneficial.

For example, the ReLAI agent can base its action choice on a precise prediction of whether or not it will collide with an obstacle in the new state $A(s_{t+1})$, where the other agent can only base its decisions on $A(s_t)$, which includes information (obstacle adjacency) that can only roughly predict future collisions between moving objects. The concrete state s_t contains enough information to predict collisions in the next state almost exactly, but this information is only useful to the ReLAI agent.

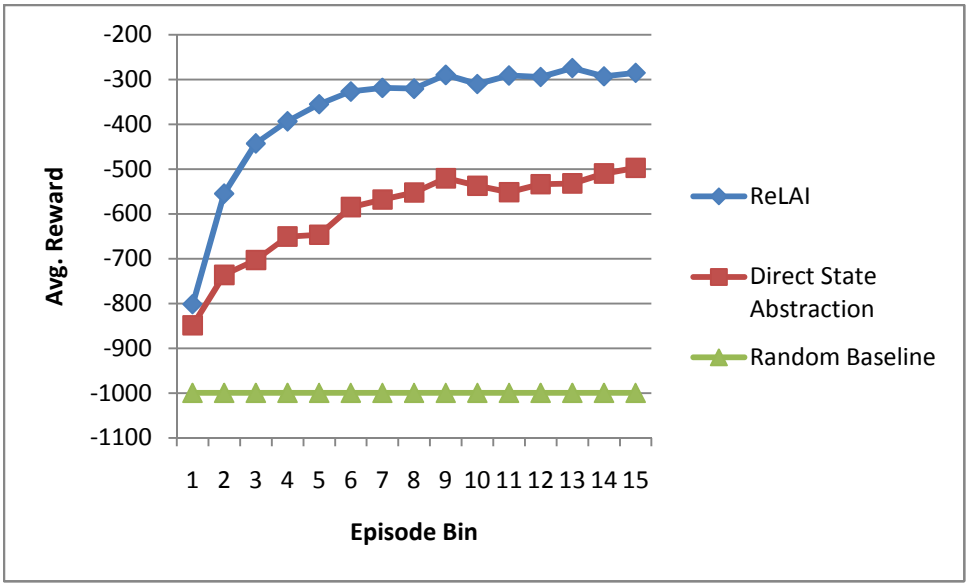

Figure 17: Performance of ReLAI vs. direct state abstraction in Frogger II.

Experiments were run using the actual (emulated) game. Q-learning with epsilon-greedy exploration was used (parameters were $\alpha = 0.3, \varepsilon = 0.1, \gamma = 0.9$). 30 trials of 6,000 episodes each were run in each condition. Figure 17 shows the results. Here, groups of 400 adjacent episodes were binned together; the results are averaged across all episodes in the bin and across all trials (each point represents 12,000 games).

As a baseline, the graph shows the estimated performance of a random agent. Random performance is generalized from data collected in 1,000 task instances, not exactly the same instances the learning agents experience. Both of the learning agents initially perform randomly, however, since they learn quickly within the first bin of 400 episodes, the graph does not reflect this.

The graphed results do not show the ability of the agents to play the game well: epsilon-greedy exploration means that the agent acted randomly 10% of the time (often with fatal results), and some of the randomly-chosen start states were unwinnable. These factors contributed to high variability in the data, necessitating the averaging of many games per data point.

To examine the final policy, 700 games were run in each condition using the final policies, but without exploration and with unwinnable games filtered out. Of these, the

Figure 18: Perceptual information in the game Space Invaders, including object labels.

direct abstraction agent received an average reward of -66 and won 45% of the games, while the ReLAI agent received an average reward of 439 and won 70% of the games.

The ReLAI agent clearly outperforms the direct abstraction agent: it learns a better policy, and learns it faster. In addition, both agents perform much better than random.

5.3.2 Space Invaders Agent

The second game addressed is Space Invaders (Atari, 1980). Figure 18 shows an example of the perceptual data provided by the emulator, overlaid with object labels provided by the low-level perception system.

In Space Invaders, the player controls a ship (located at the bottom of the figure), which can move left and right on the ground. The ship is being attacked by aliens, which drop bombs while moving left and right and (gradually) downward in a regular pattern. The ship explodes if hit by a bomb, and the player loses a life. The player can shoot missiles upward toward the aliens; its goal is to kill all of the aliens by hitting them with missiles.

All bombs and missiles take time to travel while the aliens and ship are moving – they do not arrive instantly, and the player cannot fire a missile if one is already in the air. Three static shields are located between the ship and aliens, which block bombs and missiles, but these gradually disintegrate when they are hit by missiles or bombs.

The state of the world here is continuous, since the missiles and bombs move continuously, and each instance of the game is unique, since the positions of the aliens and the timing of their bombing is different in each case.

The task addressed by the agents here is a slightly simplified version of the game, where it aims to kill all of the aliens on the initial screen (it does not progress through levels, have multiple lives, or try to kill the mothership, which is a high-value target in the full game). The agent receives a reward of 50 for killing an alien and -50 for losing a life (and ending the instance).

The state representation used encodes the following information:

-a predicate encoding a discretized horizontal position for the ship (15 possible values)

-a predicate encoding whether or not there is a "clear shot" (a missile would hit an alien if the alien stays in place)

-a predicate encoding whether or not there is an unshielded bomb (a bomb that will hit the ship if it does not move)

-a predicate encoding whether or not there a missile (shot by the ship) is in the air

-if a missile is in the air, a predicate encoding whether or not it aligns with an alien

-a predicate encoding whether or not there is a falling bomb adjacent to the ships left or right

-a predicate encoding whether or not the ship intersects a bomb

Similar to the Frogger II agents, spatial discretization is supported by augmenting low-level perception with objects representing the regions, and checking for intersections with predicate extraction. For the state variable encoding clear shots, a series of predicate extractions are necessary. For each alien, the agent queries whether or not the centroid of the ship is strictly below the alien, and whether the centroid of the ship is strictly below a shield[23]. The ship's centroid is used since the missile is fired from the center. If there is an alien above, and the shot is not blocked, a clear shot is present. Unshielded bombs and aligned missiles are similarly encoded with multiple direction queries. Adjacent falling bombs are calculated similar to adjacent objects in Frogger II, the bomb must be within approximately two ship-widths in the relevant direction to be considered adjacent.

As with the previous tasks, linear motion projection is the primary form of imagery used in this task. All aliens, missiles, and bombs can be tracked and projected forward, and the agent has been provided with motion model knowledge about how actions move the ship. However, there is one aspect of the state that cannot easily be predicted with imagery: if an agent issues a "fire" action, in the next state, a missile will be in the air (possibly aligned with an alien). This is not a simple spatial consequence of the action, since it involves a completely new object appearing. In order to simulate the effects of this action, the agent has a partially-symbolic action model for this action: it knows that issuing a fire action will cause a missile to be in the air in the next state, and assumes that missile will be aligned with an alien if there is currently a clear shot (it also knows that this will not happen if there is already a missile in the air). The use of a comprehensive architecture such as Soar easily allows this sort of integration, where symbolic action models can be used in conjunction with imagery (Laird et al., 2010).

[23] As the game progresses and the shields disintegrate, the low-level perception system can lose the ability to recognize them, as they degrade into randomly shaped objects and split into parts. The agents simply ignore shields at this point.

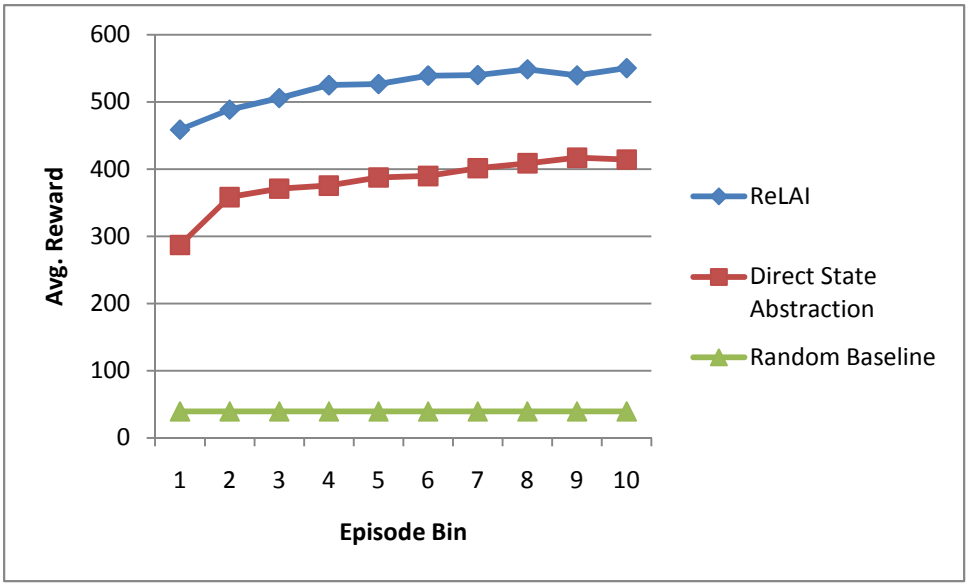

Figure 19: Performance of ReLAI vs. direct state abstraction in Space Invaders.

Experiments were run with the same parameters as in the previous section. 12 trials of 5,000 episodes each were run, groups of 500 adjacent episodes were binned together, and average reward was calculated across all trials and episodes in each bin. A random baseline was calculated by averaging the performance of a random agent in 1,000 episodes. Figure 19 shows the results.

The final policy was tested similarly to Frogger II. 720 episodes were run using the final policies, but without exploration. The ReLAI achieved an average reward of 683 (killing 13 or 14 aliens out of 36), while the direct abstraction agent achieved an average reward of 465 (killing 9 or 10 aliens). In 6 episodes, the ReLAI agent killed all of the aliens, while the direct abstraction agent never achieved that level of performance.

The performance here reflects the fact that the state representation loses relevant information from the concrete state of the problem, for both direct abstraction and ReLAI. In particular, it does not well capture the long-term motion of the objects. A direct state abstraction agent only has access to enough state information to line up its shot with the assumption that nothing will move, which does not work well here. ReLAI can do better, since it can take into account the short-term movement of objects. However, the architecture does support simulation of the complete path of missiles the agent fires, potentially allowing an imagery agent to line up its shots much more

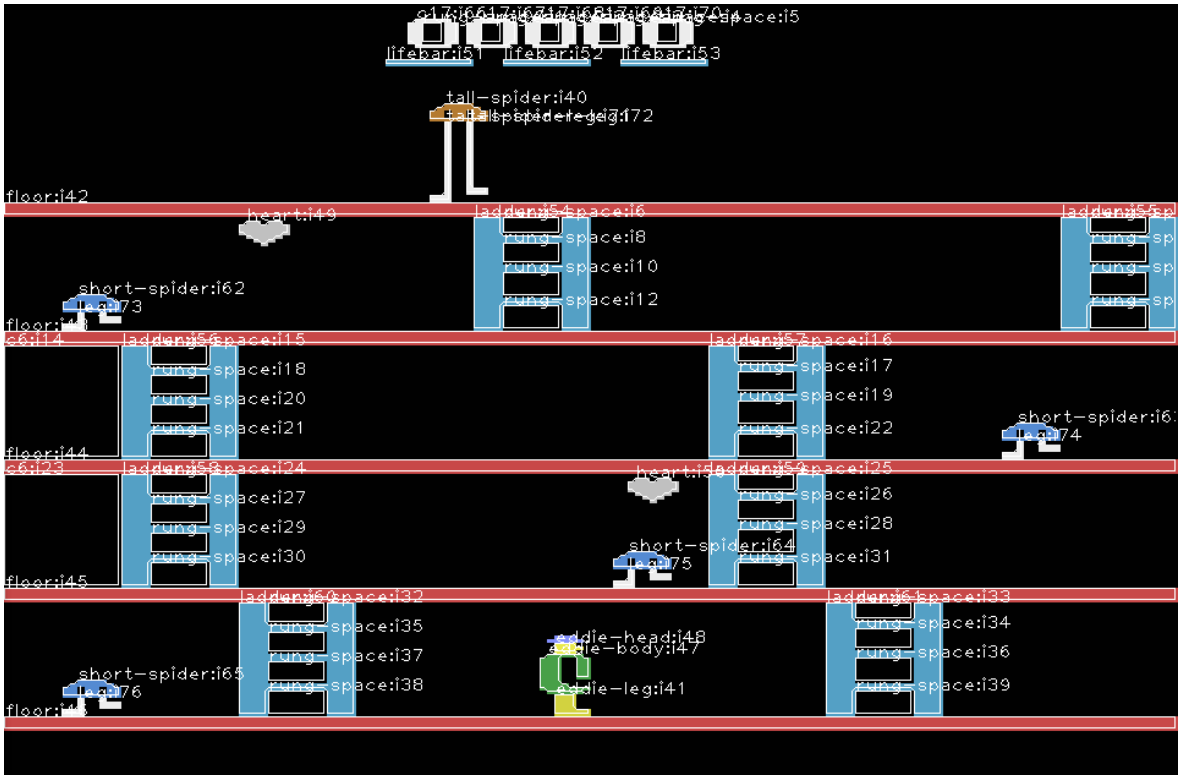

Figure 20: Perceptual information in the game Fast Eddie, including object labels.

accurately. The ReLAI algorithm, however, is constrained to use imagery only for the immediately following state. More work is needed to determine how long-term ballistic predictions that can have effects many time steps into the future can be integrated with learning.

That said, the data show a substantial advantage to using ReLAI rather than direct abstraction with the given representation, and both agents do much better than random.

5.3.3 Fast Eddie Agent

The final game addressed is called Fast Eddie (20th Century Fox, 1982). In this game (Figure 20), the player controls a character (Eddie) who navigates a two-dimensional platform world. The actions available are to walk Eddie to the left or right, jump in place, leap to the left or right (which moves Eddie roughly ¼ of the screen in that direction), climb up or down a ladder, or do nothing. There are a number of monsters in the game, which either stand still or move quickly to the left and right ("bouncing" at the edges of

the screen), and a number of heart objects that move continuously to the left or right. Eddie must jump or leap to avoid the monsters and to collect hearts. In the version of the game used for a task here, the game terminates when Eddie collects nine hearts or loses a life by colliding with a monster. Like the other games, the state space here is continuous, and each instance is slightly different.

The agent receives a reward of 50 whenever it collects a heart and -100 whenever it loses a life. For this game, the environment interface was customized to provide the agent with a set of actions available at each step, simplifying the problem so the agent does not consider moving up or down when not at a ladder or issuing actions that would cause Eddie to hit the edge of the screen.

The abstract state used for this task is as follows:

- a predicate encoding whether or not Eddie intersects a monster

-a predicate encoding whether or not Eddie is near a monster

-a predicate encoding whether or not the last action reduced the distance from Eddie to the closest heart

-a predicate encoding whether or not a heart collected was collected in the last action.

Intersections are calculated via simple predicate extraction. To determine whether Eddie is “near” a monster, distance is used. For stationary monsters, anything closer than 10 pixels (about the width of Eddie) is considered near, and for moving monsters, the distance is 35 pixels, allowing Eddie to keep a wider berth. The last two predicates are more complicated to determine and require multiple predicate extraction operations, in addition to some internal history maintenance.

To determine whether the closest heart distance was reduced, it is first necessary to find the closest heart and the distance to it. To do this, the agent determines which level of the board Eddie is currently located at, and which level each heart is located at. Rather than using special region objects as in the other games, here, predicate

extraction is used. Since the floors are objects, the agent can determine which floors a given object is above, and which floors are closest to the top of the screen. This is sufficient information to infer what floor each object is on. If there is a heart on the same floor as Eddie, the distance to that heart is used. Otherwise, the agent first infers which heart is closest *in floors* to Eddie's floor, and then determines the distance to the closest ladder adjacent to the current floor that would move Eddie towards that floor, and uses that distance. For example, in Figure 20, the closest heart is on the second floor and Eddie is on the first floor. The closest ladder to Eddie leading up is to the left of him, so the distance to that ladder is considered the distance to the closest heart. Once this distance is determined, it can be compared to that in the distance in the previous state to determine the value of the predicate. State transitions where Eddie moves between levels are handled slightly differently, but the agent is still able to accurately encode the predicate.

The direct abstraction agent can simply remember whether it has collected a heart in the last step, as that information is provided in its perceptions. However, in the ReLAI case, determining whether a heart was collected in the "last" action (which is the action being imagined) involves a series of steps. To do so, first, the agent assumes that no heart was collected unless the last action was a jump or a leap. If the last action was a leap, the agent constructs an image of the convex hull of the agent at its previous location and at its current location (using predicate projection). The agent then uses predicate extraction to check if any heart on the same floor as Eddie lies above this hull object. If the last action was a jump, the agent simply checks if a heart in the same row lies above Eddie. This process is not entirely accurate, but works in most cases. The process could be made more accurate if the detailed motion of Eddie's leaps were simulated, rather than the agent simply imagining the final state.

Both of these predicates were formulated to stretch the definition of what can be encoded in a state predicate with ReLAI. They are *backward-looking* predicates, as they encode information about the agent's recent history. These predicates can be

considered as maximally exploiting the fact that ReLAI works with state representations where the next abstract state resulting from a transition can depend on details not in the current abstract state (but in the current concrete state). Predicting future values of the predicates depends almost entirely on information completely missing from the abstract state: the locations of Eddie and the hearts. As the data (to be presented shortly) shows, ReLAI can greatly benefit from these predicates, where direct state abstraction cannot.

Compared to the other domains, the state predicates here are more complex, leading to a much smaller state space. Learning is still necessary to determine a policy; however, compared to the other domains, in this case more task knowledge is captured by the state representation. Using this representation implicitly indicates that collecting a heart, or moving closer to one, is a relevant event that should influence the agent's action choice. This property is more task-specific than simply encoding, for example, discretized locations for all of the objects.[24] However, any agent that uses an abstract representation implicitly captures properties of the task in the choice of representation: for example, segmenting raw pixels into objects implicitly captures the fact that objects are relevant to the task. This agent is simply an example from the more task-specific end of the spectrum. Investigating learning of the state representation itself is an area for future work.

A new motion model was necessary to model the monsters in this game: using the linear translation model was not accurate enough, as the monsters move quickly compared to Eddie and bounce of the edges of the screen, so in many cases a linear prediction would be wrong. The new model accounted for this bouncing behavior. Similar to the previous tasks, the agent was provided with background knowledge in a motion model so that Eddie's movement could be reliably simulated.

[24] This is a different dimension of task-specificity than that added by imagery, where the given state abstraction is applied after an imagined action, resulting in a prediction which captures aspects of the (task-specific) action.

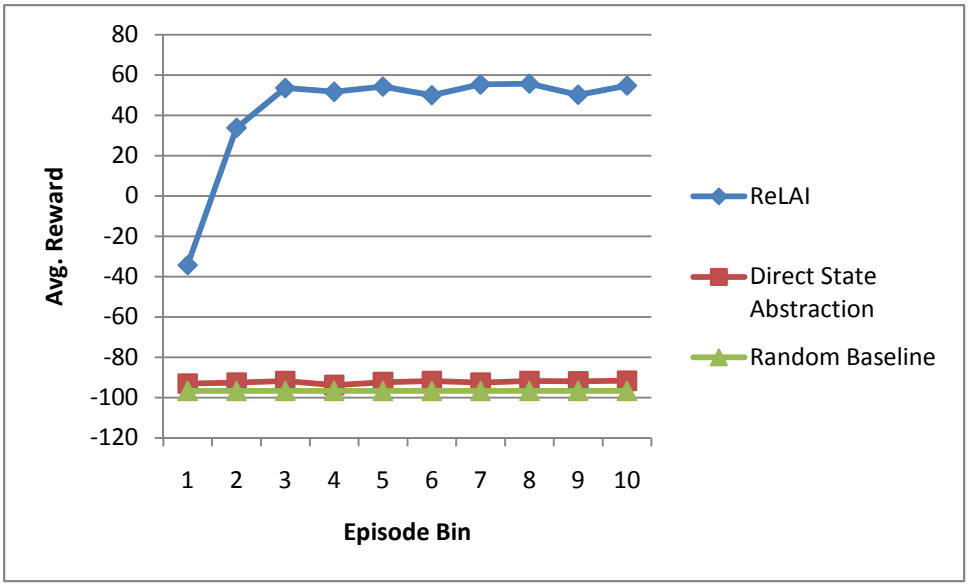

Figure 21: Performance of ReLAI vs. direct state abstraction in Fast Eddie.

Experimental data was gathered in 24 trials of 1000 episodes, and is shown in Figure 21, with average reward binned across groups of 100 adjacent episodes and all trials. Parameters were $\alpha = 0.01, \varepsilon = 0.1, \gamma = 0.9$. These are the same parameters as used in all other experiments, except the learning rate: a much lower rate (.01 vs. .3) was necessary since there are so few abstract states in the problem. To inspect final performance, 720 instances of the task were run using the 24 final policies without exploration. Here, ReLAI earned an average reward of 234, where direct abstraction earned -92. The ReLAI agent won (collected all nine hearts) in 44% of these instances.

A baseline was also determined by testing a random-action agent in 1,000 instances of the task. As the data indicate, again, in this task, ReLAI is able to perform much better than direct abstraction with the same state information and the random agent.

5.3.4 Summary of Video Game Experiments

Overall, these three video game agents all serve to demonstrate that ReLAI can be empirically useful even when theoretical requirements are not met, and that state representations that meet the theoretical requirements of neither direct abstraction nor ReLAI can perform much better with ReLAI. The predicates used in the state representations of these agents also demonstrate some of the variety of complex, task-

specific predicates that can be derived using SVS. In addition, the set of data as a whole reflects the fact that the theory and implementation of SVS is complete enough that imagery can be beneficially used in complex tasks that were not originally designed as evaluation domains for that purpose.

These agents further demonstrate the benefits listed in Chapter III. With ReLAI, the agent predicts a set of future abstract states in order to gather enough information to make a decision. In that way, an inference of the value of a predicate in a predicted future state is really an inference of a property of the current situation[25] that the agent is in. When the prediction process involves motion, then, simulative imagery is being used to capture movement in terms of abstract symbolic information (B1). When the Frogger II ReLAI agent does a one-step lookahead to infer that moving up will cause it to collide with a fish, it has inferred symbolic information that takes into account the precise movement of both the frog and the fish.

In addition, since the simulative imagery process differs in each agent based on the details of how the environment of that particular game evolves at each step and how the agent's actions cause movement, the properties of the current situation being inferred through imagery lookahead are task-specific (B2).

As was explained above, the ReLAI algorithm implicitly captures the action modeling aspect of simulative imagery (B3) where other techniques such as imagery-augmented direct state abstraction do not. Therefore, the ReLAI agents here demonstrate that benefit.

The ReLAI agents here are also strong examples of the ability simulative imagery affords of allowing decisions to be made with abstract information while predictions are made

[25] I'm using the term "situation" rather than "state" to avoid confusion with the states that play a direct role in the learning algorithm. For example, a ReLAI agent in pedestal blocks world might infer that moving **A** to **`pedestal1`** would cause a collision, which is a property of the current situation, but does not add **`collision_if_moved(A,pedestal1)`** to its RL state representation as a direct state abstraction agent would.

with concrete information (B4). The movement of objects in all three of these games is easily predictable in concrete, quantitative terms. If an object moved three pixels to the right in the last frame, it is likely to be another three pixels to the right in the next frame. This is a simple regularity in all of these tasks that makes the environments very predictable. However, including information at that level of detail in the state representation used by the RL algorithm would create far too many states for efficient learning. Using multiple levels of abstraction and imagery allows the agents to leverage low-level predictability in the environment while still maintaining a compact representation suitable for efficient learning at the decision-making level.

Chapter VI – Motion Planning in Soar/SVS

As discussed in Section 2.2, motion planning for a car-like vehicle is a challenging problem. Recall that motion planning in this case is the problem of determining a control sequence such that a robot can drive through its environment to a goal location (Figure 22).

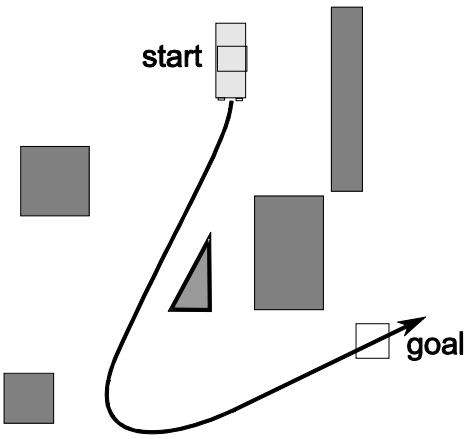

Figure 22: A nonholonomic car motion planning problem.

The difficulty here is due to the need for precise control, where the output of the agent must be sensitive to minute variations in its input. This aspect makes the problem fundamentally irreducible, as it cannot be adequately solved by choosing actions based solely on abstract states. Moreover, the most straightforward approach to handling irreducibility, the use of encapsulated controllers, is insufficient, as nonholonomic constraints make that form of abstraction very difficult.

In this chapter, an agent instantiated in Soar/SVS to address this task is introduced. This agent implements an existing sampling-based motion planning algorithm, where imagery is used to simulate the effects of a low-level controller in the current situation.

This agent provides a demonstration of the benefits in Section 3.3, most importantly, those benefits related to irreducibility, a difficulty that is not as prominent in the arcade game tasks of the previous chapter. The motion processing system of SVS is important here, as it was in the arcade game tasks, however, here the model used is much more complex, and is (theoretically) used for external control in addition to imagery.

Additionally, the algorithm used here is more complex than the reinforcement learning techniques in the previous chapter. This agent therefore provides a more comprehensive example of how high-level processing in Soar and lower-level processing in SVS can be integrated in a complete agent.

6.1 The RRT Algorithm

In response to the difficulty of abstraction in motion planning, a family of motion planning algorithms has been developed based on the principle of sampling possible trajectories through simulation. RRT (Rapidly-exploring Random Trees, LaValle & Kuffner Jr, 2001) is a sampling-based motion planning algorithm that works by constructing a tree of reachable states of the robot, rooted at the initial state, and adding nodes until that tree reaches the goal. Nodes are generated by extending the tree in random directions, in such a way that it will eventually reach the goal, given enough time. Each path from the root of the tree to a leaf represents a path that the robot could take, constantly obeying all constraints on its motion.

```
make tree rooted at initial configuration
while tree does not reach goal
  generate random configuration -> Xr
  or use goal configuration -> Xr
  with some probability
  get closest existing state to Xr -> Xc
  extend Xc towards Xr -> Xn
  if no collision occurred
    add Xn to the tree, connected to Xc
```

Figure 23: The RRT Algorithm.

The tree is constructed by the algorithm in Figure 23, and Figure 24 shows an example of one iteration of the algorithm applied to a car planning problem.

In the example, the car's current configuration is node X_0, while previous iterations have uncovered other reachable configurations $X_1 - X_4$. These configurations are linked in a tree, where each configuration is reachable from its parent via a known control. In this case, a "control" at the level of RRT is a selection of a low-level controller to use, for example, a controller that greedily steers the car toward a particular goal. The path followed by this controller between each connected configuration in the tree is shown in the figure. To add to this tree, a target configuration X_r is randomly generated, as represented in the left half of the Figure. The algorithm then attempts to extend its tree of reachable configurations to that configuration.

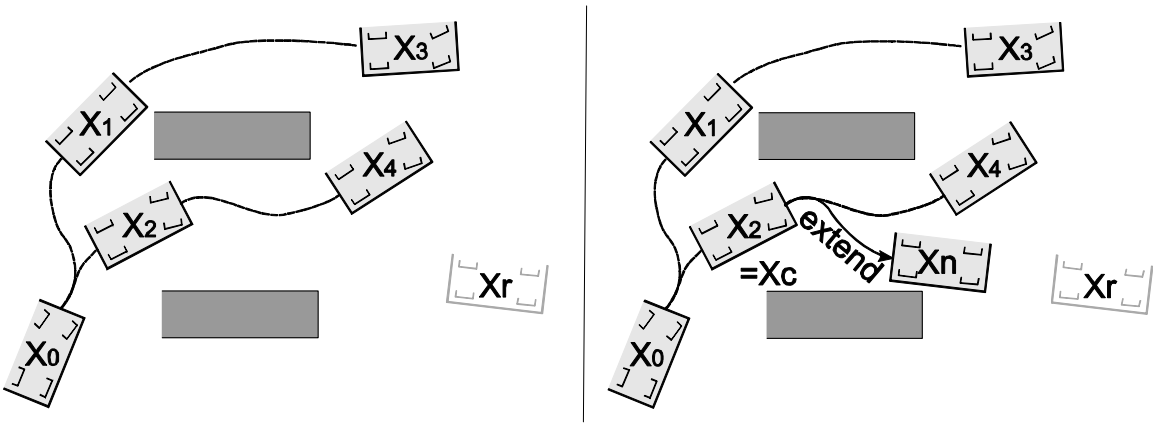

Figure 24: An example of RRT applied to car motion planning.

To extend the tree, the closest known configuration to X_r must be determined. To do this, some metric must be used that can approximate the "distance" between configurations—that is, the metric must approximate the distance of the shortest path the car could follow to move from one configuration to another. In the case of car path planning, a simple metric is the Euclidian distance between the position of the car in the two states, with the condition that the distance is infinite if the target state is not in front of the source. On the left of Figure 24, configuration X_4 is the closest to X_r given Euclidean distance alone, but since X_r is not in front of X_4, actually driving from X_4 to X_r would be difficult, since the car cannot turn in place to face X_r. X_2 is then the closest configuration to X_r once the front constraint is taken into account, and X_c in the algorithm takes on the value of X_2.

The next step in the algorithm is to extend the chosen node towards X_r, while detecting collisions along the path. This is shown on the right of Figure 24. A typical approach is to numerically integrate differential equations that describe the vehicle dynamics to simulate motion, resulting in a sequence of states parameterized by time. This simulation must occur within a system capable of detecting collisions. In the right frame of Figure 24, the controller is invoked starting at the configuration of X_2, and the car's motion is simulated driving towards X_r for some amount of time. Since no collision occurred, the new node X_n is added to the tree of reachable configurations. The algorithm then continues until the tree reaches the goal.

6.2 RRT in Soar/SVS

A version of the RRT algorithm has been instantiated in a Soar/SVS agent (Wintermute, 2009b). The problem considered is that of planning to drive a car from an initial state to a goal region, while avoiding obstacles in a known environment (the agent only determines a plan, it is not connected to an actual robot).

A complete car configuration in the version of the problem considered here consists of a position where the car is located, the steering angle, the steering velocity (since the steering angle cannot be instantaneously changed), and the angle of the car body. The

car motion model takes as input the identity of a car in the scene, and the location of a goal. By accessing the spatial scene, the model can identify the position and body angle of the car, and the other configuration aspects are initially assumed to be 0.

Inside the model, a system of differential equations describe the configuration of the car as a function of the time and goal location. When integrated, these equations can yield a sequence of configurations parameterized by time, allowing for simulation. The equations used here were determined by combining a model of human movement and obstacle avoidance (Fajen & Warren, 2003) with a simple car model (LaValle, 2006). No human modeling claims are being made with this choice of controller, rather, the particular controller was chosen as a simple demonstration of how techniques and results based on the dynamical systems approach to cognitive science can be tightly integrated with a symbolic AI framework. In addition, it performs well.[26]

The human model controls the intended steering angle of the car, and this steering angle determines the next position of the car. A constant speed is assumed. The model locally avoids obstacles: each obstacle affects the steering of the car, with nearer obstacles located towards the front of the car having the most influence. This reactive obstacle avoidance alone can solve simple problems, but more complicated problems cannot be solved this way, as a solution needs to be composed out of several distinct movement subgoals.

The controller simulates motion towards a goal, while maintaining the nonholonomic constraints of the vehicle. Along with geometric models of the car and world in the LTM of SVS, it is the low-level knowledge that was added to the existing SVS system to implement this planner.

Symbolic Soar rules were written to perform the algorithm in Figure 23. As a metric for node distance, Euclidean distance was used, with the condition that the distance is

[26] The local obstacle-avoiding controller here was directly compared (with favorable results) to a similar controller that simply steers towards the goal in (Wintermute, 2009a).

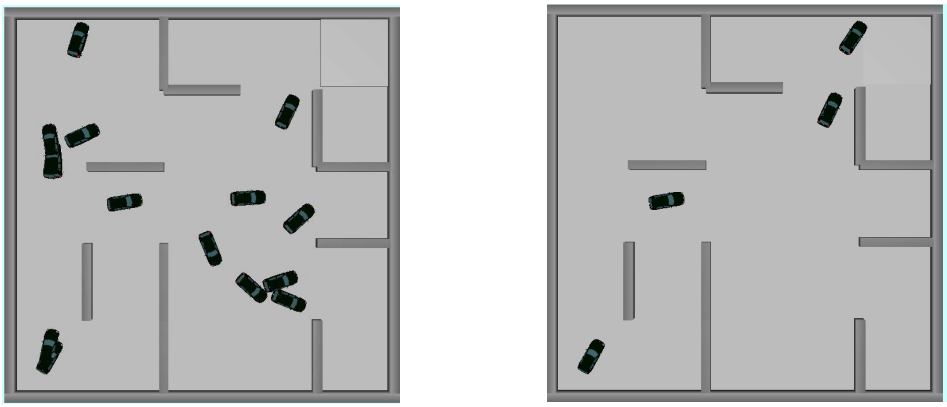

Figure 25: States of SVS Spatial Scene during RRT planning. The problem is to drive a car from lower-left to upper-right. Left: RRT tree, just before a solution is found. Right: Sequence of car positions that solve the problem.

infinite where the goal is not in front of the node.[27] SVS predicate extraction mechanisms were used to extract distances, and to query for an in-front relationship. The motion model described above enables simulation, and SVS supports querying for intersections between objects in the scene, enabling collision detection. The only new mechanism needed in SVS to support this algorithm was a predicate projection method to generate random goal points in the scene, which was a simple addition.

Examples of the SVS scene during RRT planning are shown in Figure 25. Soar stores, as a symbolic structure in working memory, the RRT tree. The nodes in that tree are perceptual pointers into SVS—they point to specific objects in the scene, which can be seen in the figure. Soar proceeds by adding a new random point object to the scene, and querying for the distance from each node to that object. These distances are then compared to find the closest. A motion model-based simulation is instantiated with that node as the initial condition[28] (creating a new car object in the scene), and this

[27] This means that the steering aspects and the body angle of randomly-generated configurations (and the goal configuration) are ignored when computing the distance metric. This allows random configurations and goals to be represented by points, rather than complete configurations.

[28] In the implemented system, motion model instantiations are not maintained after nodes are added to the tree, only the object in spatial scene remains. Since steering angle and velocity are maintained in the model (and not the scene), these quantities are assumed to reset to zero between nodes. Minor enhancements to the architecture would be necessary to allow motion model instantiations to be preserved.

simulation is stepped until a certain time is reached, the goal is reached, or Soar detects a collision with an obstacle. In all but the last case, the termination of the simulation results in a new tree node being added. In addition to moving towards random points, with a certain probability the agent instead tries to extend the tree directly towards the overall goal, biasing the growth of the tree in that direction.

The agent has been tested on the problem in Figure 25 and other similar scenarios. For the problem in the figure, 100 trials were run, and a solution was found after an average of 12 tree expansions. However, the primary purpose of this agent is to serve as an existence proof that the algorithm can be implemented in Soar/SVS, and as a demonstration of the architecture applied to this task. For those purposes, experimental data beyond stating that the implementation functions as described is largely redundant.

```
 1: O: O22 (svs-build-world)
    ...
 9: O: O291 (rrt-plan)
10: ==>S: S10 (operator no-change)
11:    O: O293 (init-tss)
12:    O: O294 (generate-projection)
13:    O: O300 (wait-for-svs)
14:    O: O301 (rrt-extend)
15:    O: O303 (run-svs-simulation)
16:    ==>S: S12 (operator no-change)
17:       O: O343 (svs-step)
          ...
37:       O: O363 (svs-step)
38:    O: O365 (register-node)
39:    O: O366 (generate-projection)
40:    O: O371 (wait-for-svs)
41:    O: O372 (rrt-extend)
42:    O: O374 (run-svs-simulation)
43:    ==>S: S14 (operator no-change)
44:       O: O414 (svs-step)
          ...
59:       O: O429 (svs-step)
60:    O: O431 (remove-failed-sim)
61:    O: O432 (generate-projection)
62:    O: O437 (wait-for-svs)
63:    O: O438 (behind-all)
64:    O: O439 (generate-projection)
65:    O: O444 (wait-for-svs)
66:    O: O445 (rrt-extend)
67:    O: O447 (run-svs-simulation)
68:    ==>S: S16 (operator no-change)
69:       O: O487 (svs-step)
          ...
82:       O: O500 (svs-step)
83:    O: O502 (register-node)
84:    O: O503 (generate-projection)
85:    O: O508 (wait-for-svs)
86:    O: O509 (rrt-extend)
87:    O: O511 (run-svs-simulation)
88:    ==>S: S18 (operator no-change)
89:       O: O551 (svs-step)
          ...
99:       O: O561 (svs-step)
100:   O: O563 (remove-failed-sim)
101:   O: O567 (choose-goal)
102:   O: O569 (rrt-extend)
103:   O: O571 (run-svs-simulation)
104:   ==>S: S20 (operator no-change)
105:      O: O611 (svs-step)
106:   O: O613 (remove-failed-sim)
107:   ... etc.
```

Figure 26: A trace of a Soar/SVS agent executing RRT planning. Initialization is shown in blue, iterations of the algorithm are shown in alternating font styles.

6.3 Symbolic Soar Processing in RRT

To provide an example of how higher-level Soar processing interacts with SVS, Figure 26 shows a trace of Soar's behavior during RRT planning.

Symbolic Soar processing is mediated through a large declarative symbolic structure, the working memory. Reasoning in Soar is accomplished through a sequence of *decisions*, where an operator, representing a specific choice of an internal or external action, is selected. Symbolic rules control which

operators are available at what time, choose particular operators when there are multiple possibilities, and change the state of working memory based on the selected operator. Subgoaling in Soar is achieved via impasses, which arise whenever there is insufficient knowledge to immediately choose an operator.

Each line in the figure shows the name of the operator chosen at that decision, or that an impasse was entered (indicated by a ==>). More details about symbolic Soar processing can be found in (Lehman et al., 2006), and more detail about how SVS interacts with Soar's working memory can be found in (Wintermute, 2009b).

Each iteration of the algorithm starts with a decision to either generate a random point to use as a goal (seen as a **`generate-projection`** decision), or a decision to try and extend the tree towards the overall goal (seen as a **`choose-goal`** decision in the last iteration). After deciding on the location to extend towards (a target), the agent must determine which existing state (which node in the tree) should be extended toward that state. This involves predicate extraction queries: orientation queries are set up so that the **`in-front-of`** relationship is checked between each node in the tree (which is a pointer to an actual car object in the scene) and the target object, along with distance queries between those. From Soar's perspective, these queries are performed in parallel for all objects. Of all of the nodes for which the orientation query matches, that with the shortest distance is chosen. If a random point is generated that is not in front of any node, it is rejected, this is the **`behind-all`** operator in the third iteration. The **`wait-for-svs`** operator selected at several points in the trace is an artifact of the way the current implementation of SVS interfaces to working memory and is unimportant for this discussion.

After finding a node to extend, the agent uses a motion model to simulate the car driving towards the target. This is set up through the **`rrt-extend`** operator. SVS includes task-independent library rules to allow easy creation of motion simulations. Here, the agent specifies to simulate moving the car towards the target, until either a maximum time is reached (the common case for a successful extension), or a predicate

extraction query matches, detecting that the car has collided with an obstacle or reached the goal. As the motion model used avoids obstacles, the perceptual pointers to the obstacles in the problem must also be provided to the SVS motion simulation process. After the relevant queries are set up, the operators **`run-svs-simulation`** and **`svs-step`** (part of the SVS library) take over. Based on the results of the simulation, the image of the car at its new location is either added as a new node (if the extension was successful), or removed from the scene (if a collision occurred). This continues until the overall goal is reached, at which point the agent removes all of the non-solution nodes in the tree (as seen on the right of Figure 25). The agent halts at this point. In a more complete agent, the plan would then be executed by moving the robot towards these locations.

6.4 Perceptual Abstraction and Irreducibility in Motion Planning

Now that the agent has been described, it is worth revisiting the principles introduced the earlier chapters, to see how this system demonstrates the benefits of the theory.

First, it is important to note that the algorithm here was developed by other researchers, completely independent of any broad architectural theories, but instead to solve a practical engineering need. That indicates that the approach is fundamentally valuable, its utility is not, for example, an artifact of the symbolic assumption in Chapter II, nor of any shortcomings of the Soar architecture. While the algorithm was not originally described in terms of multiple representations and imagery, it easily maps on to those concepts. Any system implementing RRT in problems such as this requires both the means to simulate action in terms of low-level information, and to make abstract judgments about the outcome of that simulation, such as “collided with an obstacle” or “reached the goal”. In addition, information about the state of the search needs to be maintained at multiple levels of abstraction: the agent needs to maintain the exact quantitative values for each configuration in its tree, but also the abstract knowledge about the topology of the tree itself (which configurations are reachable from which others).

This agent serves as a demonstration of the particular benefits outlined in Section 3.3. Most prominently, it demonstrates the benefits related to irreducibility. Motion planning, in general, is an irreducibly large problem. If the problem were to be addressed by a purely-symbolic system, where raw perceptions were abstracted into states and mapped to raw actions, even for simple robots, the maximally abstract state space would be extremely large. However, a concrete controller is used in this approach, allowing the action space of the symbolic part of the agent to be simplified, and working to mitigate irreducibility (B5).

Concrete control alone is insufficient to mitigate irreducibility in this situation, though. The task cannot be reduced to abstract symbolic reasoning about concrete control (the encapsulated controller approach), as may be possible in simpler motion planning problems. For the nonholonomic motion here, there is no simple geometrical property of the obstacles that could be calculated to determine a small set of reachable locations and paths a controller could follow that could be searched over, as may be possible when planning the motion of simpler robots (e.g., a visibility graph (Lozano-Pérez & Wesley, 1979)). It is possible to build a conservative abstract map of the world, if there are regions that are clearly traversable, but solution quality would be lost.

Instead, simulative imagery is used in this agent to allow reasoning about the controller in terms of symbolic information, but without requiring a complete symbolic characterization of the controller (B6). This allows for a controller that entails a complex interaction with the world—steering that continuously varies with the exact positions of the obstacles and goal—to be reasoned about in terms of extremely simple abstract information: whether or not the car collides with an obstacle.

The system can also be viewed in terms of the perceptual abstraction related benefits. The use of simulative imagery allows motion to captured symbolically (B1), allows task-specific abstract properties to be captured (B2, for example, “if the controller is used to seek towards the goal from this state, it will succeed”), and allows action modeling (B3), since the agent knows that the result of a particular simulation is a property of a

particular action choice (controller selection) in the current state, rather than just another property of the current state.

Chapter VII - Related Work

This thesis touches on many areas of research in artificial intelligence, cognitive science, and robotics. This chapter contains a survey of related work, with an emphasis on systems that have influenced the theory and broad architectural design, rather than prior work related to particular aspects of the implementation or experiments (some of which has been noted in previous chapters).

Related work is divided into rough categories; however, some of the approaches described here could fit in multiple categories.

7.1 Qualitative Spatial Reasoning

Much work in the past has focused on attempts to build useful abstract representations of spatial information. Researchers in Qualitative Spatial Reasoning study this problem. An influential system here is the MD/PV model of Forbus (1983). A major result of this line of work, the poverty conjecture, was discussed earlier. Beyond this, the model itself provides some useful insights. It consists of a Metric Diagram (MD) and Place Vocabulary (PV). The metric diagram is similar to the spatial representation in SVS, it typically contains a set of labeled objects represented quantitatively. This diagram is used to build a task-specific place vocabulary, which consists of "contiguous regions of space where some important property (e.g., in contact with a gear, inside a well) is constant". Systems can be studied by looking at their dynamics in terms of the place vocabulary, for example determining if two balls moving through space could possibly collide by looking at their trajectories through the places. In a large project, the framework was successfully used to model the mechanisms of a clock (Forbus et al., 1991).

A place vocabulary can be viewed as an abstract state space, and since the metric diagram is used to generate the place vocabulary, the approach is very similar to the

imagery agents used here. However, the MD/PV model has been applied in very different tasks than those addressed here. In general, qualitative reasoning work has focused on tasks like understanding a large system: for example, predicting that a bouncing ball could eventually wind up in a pit. These are very different from the spatial control tasks addressed here. In addition, MD/PV is not proposed in detailed architectural terms: there is no comprehensive proposal for what task-independent representations and mechanisms are necessary for an agent to use the MD/PV approach in arbitrary tasks it may encounter.

In spite of the poverty conjecture, work in qualitative representations of space that can be used in a task-independent manner has progressed. Useful systems have been developed, even if they cannot capture every important interaction in every domain. As discussed in Section 4.4, the direction relationships that can be queried in SVS are based on the "acceptance area" approach of Hernandez (1994). Previous versions of the system (Wintermute & Laird, 2007) have allowed the agent to extract topological relationships based on the Region Connecting Calculus (Cohn et al., 1997); the current version does not support a complete set of relationships, but this is something likely to be added in the future. Qualitative representations like these are usually discussed in the context of formal methods for reasoning with that information, which are not used in any of the SVS agents developed so far (although they could be implemented as knowledge in Soar).

7.2 Cognitive Architectures

There have been several previous cognitive architecture approaches toward addressing spatial tasks.

7.2.1 Other Soar Extensions

The most relevant existing system to SVS is SVI (Lathrop, 2008; Lathrop & Laird, 2009). SVS inherits much of its design and code from SVI, so it is difficult to say precisely whether or not they are different systems. All of the agents developed for SVI could theoretically be adapted to SVS, but this has not been done, as the interface to symbolic

working memory has changed substantially. The motivation behind the design of SVI is to "…explore the utility of general-purpose, intelligent systems supporting mechanisms to encode, compose, manipulate, and retrieve symbolic and perceptual-based representations" (Lathrop, 2008). The basic structure of the system is the same as SVS: it has short-term and long-term memories for visual and spatial information, and means by which symbolic processing can use them. In general, SVI was more directly inspired by psychological theories of imagery (Kosslyn et al., 2006), while SVS emphasizes increasing functionality, but both systems are concerned with both areas to some degree. Work in SVI also emphasized computational efficiencies of depictive representations for visual imagery, while work in SVS has addressed the broader interaction between abstract and concrete (typically spatial) representations.

Architecturally, the chief differences between the systems are in the interface between perceptual and working memory. SVI's equivalents to the predicate extraction, predicate projection, and memory retrieval systems in SVS are simpler, and SVI has no direct equivalent to the motion processing system in SVS, either for imagery or control. SVI also has a different approach to interfacing with Soar's working memory, which is elaborated in a technical report (Wintermute, 2009b).

SVS also inherits substantially from SRS (Wintermute and Laird, 2007, 2008). Again, the chief theoretical components of the system remain intact, although some older agents would need to be rewritten due to interface changes. SVS is more directly simply a newer version of SRS, so it will not be covered in more detail here.

Another extension to Soar for spatial processing, BiSoar, has been created by Chandrasekaran and Kurup (Chandrasekaran, 2006; Chandrasekaran & Kurup, 2007), augmenting Soar with the functionality of a diagrammatic reasoning system, DRS (Chandrasekaran et al., 2004). This system is similar to Soar/SVS in many ways. BiSoar focuses on processing with two-dimensional diagrams, which are a similar representation to the spatial scene of SVS, consisting of labeled objects in a quantitative representation. In BiSoar, the state of the diagram is conceptually an extension of

working memory, as in SVS (although, in both cases, the systems are external to Soar and connected over the i/o links). The integration of spatial and symbolic state conceptually differs in BiSoar, though, as it has been proposed to include matching against spatial objects in rules, a capability which remains unimplemented. SVS instead commits to matching of qualitative properties of spatial objects, rather than the objects themselves. BiSoar has been used to model simplification effects (loss of detail) in the storage and recall of spatial memories, a capability SVS lacks (as it currently lacks means to store new long-term perceptual memories) (Kurup & Chandrasekaran, 2007).

The equivalents to SVS's predicate extraction and projection processes in DRS (and presumably also in BiSoar) are also notably different. In DRS, routines can be defined by providing a logical description of the desired property or diagram object. For example, an image of a region behind a curve object **c** relative to a point object **p** can be created by the system, based on a formal definition of the region similar to "the region contains all points **q** such that there is a point **a** which is on **c** between **q** and **p**". This statement can be broken down to primitives and automatically solved, allowing the system to compute the region (Banerjee & Chandrasekaran, 2007). Extraction predicates can be similarly described, which simply return true or false instead of creating a new object in the diagram. In contrast, in SVS, the focus has been on providing a small library of useful predicate extraction and projection routines, and allowing the agent itself to compose them together to get more complex properties, if needed.

BiSoar also has no direct equivalent to the motion processing system of SVS, and does not include three-dimensional processing.

ADAPT (Adaptive Dynamics and Active Perception for Thought) is a robotics architecture based in part on Soar that includes specialized spatial processing (Benjamin et al., 2004, 2006). This processing is chiefly used in the aide of comprehending sensor input. A world model, similar to the spatial scene in SVS, is used, where the agent builds a representation of its current hypothesis about the contents of the world. For example, if it has evidence from sensors that it is in front of a chair, it will imagine a chair. This

model is used to confirm or rule out hypotheses about the world, by checking if further sensor input is consistent or inconsistent with the current imagined scene. Interestingly, this spatial representation is not connected to sensors (as has been proposed for SVS), but rather is connected only to Soar, which mediates all sensor data and chooses what to imagine in the world model, reflecting the designers strong commitment to active perception. The world model is also proposed to be used for "comprehension through generation", where potential future states of the world are simulated in order to comprehend the current state. This is essentially simulative imagery, but it is unclear precisely how and to what degree the capability is implemented.

7.2.2 ACT-R

The ACT-R cognitive architecture includes visual processing in its current form (Anderson et al., 2004). In addition, two proposals have been put forth to augment ACT-R with spatial representations (Gunzelmann & Lyon, 2006; Harrison & Schunn, 2003). In general, research in ACT-R emphasizes precise predictions in terms of human performance and timing, and the architecture reflects this. Where Soar has a symbolic working memory of unbounded size, ACT-R allows only a relatively small amount of information to be accessed by the central production system, contained in a few buffers specific to particular cognitive processes: for example, memories retrieved from long-term memory are placed in the declarative retrieval buffer, and only one such memory can be there at a time. These limitations often drive timing predictions.

ACT-R includes an "imaginal" buffer; however, this buffer is not used for imagery as defined here (i.e., aspect A3 in Chapter III). Rather, this buffer is used to store temporary problem state information (Anderson, 2005). Like all buffers in ACT-R, the information stored there is propositional, and typically abstract. While the argument for the use of imagery here has focused on the bimodal aspects of imagery, given constraints on the capacity of working memory, the use of the memories in SVS as temporary scratchpads could also become valuable, reflecting use similar to ACT-R's imaginal buffer.

The ACT-R visual system provides a simple symbolic interface to the central production system representing 'where' (through a visual location buffer) and 'what' information (through a visual object buffer). These modules mimic the timing of human perception, and are based on the EPIC cognitive architecture (Kieras & Meyer, 1997). This emphasis on timing results in a visual system that is not directly similar to that in SVS, which has no attention mechanisms, and instead focuses on the internal representation of visual data.

ACT-R/S (Harrison & Schunn, 2003) adds two more systems to ACT-R's visual processing, the manipulative and configural systems. The manipulative system represents detailed 3d information about the objects near the agent (which it could potentially manipulate), while the configural system represents the approximate positions of objects in space relative to the agent. Roughly, the configural and manipulative systems here correspond to the object and transformation information encoded in the spatial scene of SVS. At least the configural system of ACT-R/S has been demonstrated to perform a version of imagery; however, nothing published about the system has discussed imagery in the manipulative system, which would be closer to the type of imagery used by SVS.

The proposed extension of Gunzelmann and Lyon (2006) similarly defines a number of new modules: a visual egocentric buffer, which encodes information about objects relative to the position of the agent, a visual environmental buffer, which encodes locations of object relative to environmental landmarks, a spatial buffer and associated module which supports querying processes similar to predicate extraction, and an episodic buffer, which captures cohesive memories of the agent visual experiences. This proposed system would support imagery. In the theory, memories can be retrieved in the episodic buffer, and spatial imagery transformations are possible: production rules can manipulate the egocentric buffer, and the spatial module will propagate those changes to the visual object and location buffers. These processes are roughly analogous to the memory retrieval and predicate projection systems in SVS. However,

the architecture here does not have a clear distinction between concrete and abstract representations, so it is difficult to draw a deep comparison between the systems.

7.3 Robotic Systems

Systems developed to support intelligent robotics often address many of the issues discussed in this thesis. For example, the system developed for MIT's entry in the DARPA Urban Challenge (Leonard et al., 2008) was referred to earlier as an example of real-world use of the RRT motion planning algorithm discussed in Chapter VI. This system can be viewed as having most of the aspects listed in Chapter III. It represents the state of the world at multiple levels of abstraction (A1): a Navigator module choose routes based on lanes, intersections, and road segments, while a Motion Planner module takes goal locations from the Navigator and performs RRT planning over a more concrete "drivability map" representation. As explained above, RRT planning can be viewed as a form of simulative imagery (A4). The system included low-level controllers for driving the car (A5), and the system uses those same controllers offline during motion planning (A8). While this system, like many others, implements aspects of the theory, it not proposed nor implemented as a general-purpose architecture, and these aspects are considered engineering details rather than theoretical commitments. The remainder of the related work here will focus on robotics systems that are posed more generally.

The Spatial Semantic Hierarchy (Kuipers, 2000) presents a comprehensive theoretical treatment (and implementation) of robot navigation, with a focus on mapping. In its most recent incarnation (Beeson et al., 2010), the system includes four main representational levels, containing both metrical and topological information about both small-scale space (space within the range of the agent's sensors) and large-scale space, all of which are algorithmically constructed from sensor data. In this system, the connections between low-level control and high-level representation are explored in detail. Control laws are used which can reliably transition the robot between distinctive states, and, at higher levels of the hierarchy, the control laws are abstracted away and

the agent only considers moving between distinctive states: this is the encapsulated controller approach discussed in Chapter II.

Conceptually, the metrical representations of space could be mapped onto SVS's spatial scene, and topological spatial information could be incorporated in Soar's symbolic working memory. More study would be needed to determine what would be needed for Soar/SVS to implement the details of the higher levels of SSH, though. While the SSH does not use imagery in the sense defined here, it does share a commitment to representation at multiple levels of abstraction and hierarchical control, corresponding to aspects A1 and A5 of the theory here.

Other robotic systems have previously implemented capabilities that can be considered simulative imagery. MetaToto (Stein, 1994) was a robot designed based on Brooks' subsumption architecture (Brooks, 1986), which used simulation in order to derive abstract information about the structure of the world. These simulations were at a very low level, the actual sensor readings of the robot were simulated (in contrast to SVS, which simulates in a higher-level spatial representation). The robot represented the world in terms of landmarks corresponding to distinctive sensor readings, and by simulating sensor readings based on a map of the world, it could build this representation without actually exploring.

The robot here represents the world at multiple levels of abstraction (in terms of sensor data and a "landmark graph") (A1), and has controllers that operate over the lowest-level sensor data (A5). The agent is able to derive the abstract consequences of using these controllers (landmarks reached) through low-level simulation (A6). There is also a strong commitment to the reuse of the perception and action systems for real and imagined situations (A8).

The theory behind MetaToto is also interesting. Stein claims "our view suggests that cognition is simply the robotic architecture applied to imagined stimuli". Certainly, this form of reasoning is powerful, but in SVS cognition is by no means "simply" reduced to simulation: abstract knowledge and processes also play prominent roles.

Two general-purpose robotics architectures, 4D/RCS (Madhavan et al., 2006) and Polyscheme (Cassimatis et al., 2004) include the capability for functionality like simulative imagery. In both cases, though, the architecture is considered in very different terms than that presented here, as a set of many interconnected specialized modules. There are fixed commitments to the way modules connect, and to some degree, the internal processing. But there are not commitments at the level of, for example, a common task-independent language to communicate between spatial and symbolic information, as in Soar/SVS. In addition, simulative imagery is not heavily analyzed in the context of these architectures as a distinct process. Rather, it is considered one of many ways that predictions about the consequences of actions can be generated.

7.4 Reinforcement Learning

Previous work in the area of reinforcement learning has often examined the problem of learning and control in problems with large state spaces. As was discussed previously, spatial information is inherently continuous, often entailing very large state spaces. One approach to this issue is to use qualitative abstractions of the low-level spatial state and induce an abstract state space. This approach is used in Chapter V and in other work (e.g., Stober & Kuipers, 2008).

However, other approaches to dealing with large spatial state spaces have been investigated. Often, continuous information is not abstracted from the states of the agent, and rather than learning unique action or state values, the agent instead tries to learn a function over the state elements which approximates the values[29]. In the resulting system, an agent experiencing a completely new state can essentially leverage knowledge learned in other states that are "nearby" in terms of the values of the state.

[29] Formally, state abstraction and state-action abstraction can be viewed as special cases of function approximation, following the definition in (Sutton & Barto, 1998). However, we will treat the approaches as separate here, since these abstractions have interesting properties missing from general function approximation (such as the ability to generate a reduced MDP in the case of state abstraction, as discussed in Chapter V).

A common approach to function approximation is to use sparse coding mechanisms, such as CMAC (Sutton, 1996). CMAC overlays different tilings (discretizations) over space, where any particular location will match multiple tiles. Values are learned in terms of the tiles matched at the time of the update, so reward information learned about a given concrete state (e.g., one represented in continuous coordinates) will influence the values of states around it. Function approximation methods like tile coding can be integrated with reinforcement learning in Soar (e.g., Wang & Laird, 2010), and SVS could be used to obtain qualitative tiling information; for example, "the agent is to the right of object X and in front of object Y" describes a location in two tilings.

Other approaches to function approximation use more complex means of learning a value function, rather than simply combining values associated with sets of overlapping features. For example, a neural network can be used to learn a complicated relationship between state variables and values, as has been used in a successful agent for the game backgammon, TD-Gammon (Tesauro, 1995, see also Sutton & Barto, 1998 ch. 11).

There is a concern here, since the motivation of the architectural design presented here has to do with the perceptual abstraction problem. If function approximation schemes successfully deal with large state spaces without the need for explicit abstraction, it may be that good function approximation supersedes the benefits of imagery for dealing with large state spaces.

This does not seem to be the case, however. For example, the TD-Gammon agent cited above uses processes that can be viewed as simulative imagery in conjunction with function approximation. In that agent, when considering each move, the agent first determines the consequences of that move in terms of a low-level game board representation. Then, for the resulting state, abstract features of the game board are calculated. These features (along with the board state) are the inputs of the neural network that approximates the value of the state. This behavior fits the description in Chapter III of bimodality (A1) and simulative imagery (A4), but also incorporates function approximation. As TD-Gammon is the result of a long line of research,

presumably all of these aspects are important for its performance, and it can be concluded that simulative imagery and function approximation are at least partially complementary.

7.5 Grounded Cognition

A source of influence in the design of the architecture has been theories of grounded cognition. Barsalou (2008) states:

> "Grounded cognition rejects traditional views that cognition is computation on amodal symbols in a modular system, independent of the brain's modal systems for perception, action, and introspection. Instead, grounded cognition proposes that modal simulations, bodily states, and situated action underlie cognition."

Of course, this point of view is not completely reflected in the architecture here. Indeed, the architecture is rooted in Soar, which is a prominent example of the amodal symbolic view of cognition that is "rejected" by these theories (e.g., Newell, 1990). However, the system retains and uses perceptual-level information during cognition, with the perception and action systems internally used in this process. In that way, the system bears a resemblance to theories such as Barsalou's (1999) proposal for a perceptual symbol system and Grush's (2004) emulation theory of representation.

The implementation has shown that the existing symbolic Soar architecture is compatible with non-symbolic processing in SVS. This is because Soar is largely a theory of high-level decision making, complementary to the processing in SVS that identifies properties and makes inferences but does not choose actions or control high-level processing. From this point of view, information in Soar's working memory can be considered as symbolic *aspects* of underlying representations in the memories of SVS. Specifically, perceptual pointers can be viewed directly in this way: a perceptual pointer can be manipulated like any other symbol, but retains its underlying non-symbolic structure. However, other symbols are used in the system that are not grounded in perceptual representations (amodal symbols).

For example, the pedestal blocks world agent discussed in Section 5.1 makes a series of imagery-based predictions about whether or not each action will result in a collision between blocks (e.g., **`collision_if_moved(B,pedestal2)`**). As implemented, for each possible action, the agent imagines the state resulting from a block movement, and constructs a symbolic structure to store the results of the simulation (encoding whether or not a collision occurred). Importantly, after storing this symbolic result, the underlying perceptual-level representation is no longer necessary, and is removed. As no equivalent perceptual-level structure exists, the symbolic prediction is now amodal, and does not meet Barsalou's (1999) description of a perceptual symbol structure. Once predictions for all actions are computed (up to six of them), the agent uses this set of symbolic information as the basis to choose an action. This is a relatively small set of symbolic structures, encoding only the relevant aspects of the situation.

However, if all symbolic structures were required to be perceptually grounded at all times, the agent would need to retain, until the action is chosen, the complete perceptual-level detail that caused each abstract symbolic prediction result to occur. This agent would then be required to represent all possible action outcomes *simultaneously* in imagery. In general, as they lack in unnecessary detail, amodal (ungrounded) symbols are more efficient means of representing information. While this agent benefits from representing *some* information at the perceptual level (supporting the main claims of this thesis), requiring *all* information to have grounding at that level would be a hindrance.

There is then a substantial functional benefit to avoiding the use of perceptual-level representations when amodal symbolic structures are sufficient to capture all of the necessary details of the situation. In Soar/SVS, the processing necessary to manipulate and use amodal symbolic structures is a subset of that needed to manipulate and use grounded structures, so a compelling argument would be needed to eliminate this capability.

Related to discussions of the role of amodal versus grounded representations, connections between this work and the imagery debate in psychology will be discussed in the next chapter.

Chapter VIII - Discussion and Conclusion

In this chapter, the claims put forth in earlier chapters are revisited and discussed in light of the overall collection of experimental results. Then, connections between this work and psychological research are discussed. Future directions for research are indicated, and the main body of the dissertation is concluded.

8.1 Claims Revisited

The overall goal of this thesis has been to work towards a task-independent cognitive architecture to support intelligent behavior in spatial tasks. Starting from an assumption that abstract symbolic information is used to choose actions, in Chapter II, two meta-problems were defined that the architecture must address: the problem of creating appropriate abstract symbolic structures which can serve as the basis for intelligent action choices (perceptual abstraction), and the problem of dealing with tasks where abstract, purely-symbolic representation is impossible (irreducibility).

To mitigate these problems, a comprehensive theory was proposed in Chapter III. Three high-level claims were made about this theory: that it allows for improved performance in individual spatial tasks, that it allows for improved generality, and that it can be practically implemented and used. The theory was mapped to eight specific functional benefits, each of which supports one or more of the claims, and relies upon a particular architectural aspect. Performance or generality improvement was claimed compared to a hypothetical similar architecture lacking that aspect.

The Soar/SVS architecture, which follows the theory, was introduced in Chapter IV. Several agents running in this architecture were introduced. In Chapter V, reinforcement learning agents were presented which were applied to a simple blocks world problem and three arcade games, and in Chapter VI, a motion planning agent was covered.

In this section, the claims and benefits are re-examined in the light of the collection of experimental results presented since their introduction.

B1. Concrete routines allow movement and nonlocal interaction to be captured in terms of abstract symbolic information, mitigating the perceptual abstraction problem.

This benefit is demonstrated in all of the agents. In the pedestal blocks world, nonlocal interactions (block collisions in future states) are captured explicitly as predicates in the imagery-augmented direct abstraction agent, and implicitly in the predictions of the ReLAI agent for the same task. In the arcade game agents, in all cases, movement of multiple objects is captured by the ReLAI agents, again, implicitly via the predictions of future states. In the RRT planning agent, the movement of the car under the influence of the low-level controller is captured symbolically, as the tree of configurations only includes those that a reachable without collision, a property deriving from the details of that movement.

The imagery-augmented abstraction and ReLAI agents, in their comparison to the direct state abstraction agents, also provide direct evidence that capturing movement truly is a benefit. When the same state information is used, but movement is not captured via either technique, the agents perform much worse.

This benefit is attributed to concrete routines in the architecture: the ability for a concrete representation to be locally manipulated. Here, these concrete routines are invoked through imagery, although that aspect is not strictly necessary for the benefit. These agents provide good examples of properties that would be very difficult to capture without this aspect. For instance, the reachability of two configurations in the RRT planning agent is determined here through a long concrete simulation process. If there was no coherent concrete representation that could be locally manipulated in this way, it is difficult to see how the agent could infer this long-term reachability information.

B2. Imagery allows task-specific abstract properties to be encoded by a fixed, task-independent high-level perception system, mitigating the general perceptual abstraction problem.

This benefit is demonstrated by the set of examples, both individually and collectively. The SVS architecture in Chapter IV includes a fixed, task-independent high-level perception system, which all of the agents presented use to capture task-specific abstract properties. In each case, the imagery system is dynamically combined with the high-level perception system to generate properties that take into account the spatial details of the actions available in the particular task.

In addition, the theoretical examination of ReLAI provides further evidence that this capability truly works to mitigate the general perceptual abstraction problem. A high-level perception system that cannot induce an abstraction function to meet the formal requirements of direct state abstraction might be able to induce such a function that works with ReLAI. For example, SVS's predicate extraction system cannot induce an abstraction of pedestal blocks world that allows optimal performance with direct state abstraction, but can induce an abstraction that works with ReLAI. Imagery capability has thus increased the coverage, in terms of number of tasks, of a particular high-level perception system, achieving progress towards general perceptual abstraction.

This benefit is attributed to imagery, rather than concrete routines alone, since, as was stated in Section 3.3, "the full reasoning power of the agent must be brought to bear to select which concrete routines to apply and how to interpret or further manipulate the results". This is demonstrated by all of the agents. In each case, symbolic Soar rules have been created which specify exactly how the agent should manipulate the spatial scene via imagery, and how the results should be interpreted, via predicate extraction, into abstract properties. If concrete routines alone were to be used for similar capability, task-specific knowledge would need to be present somewhere within the perception system of the agent, so the agent could apply different sequences of routines as appropriate in different tasks.

Even if such a scheme could be created to allow task-specific sets of concrete routines to be used, learning would be a problem. Since the rules that specify these operations in the approach here are essentially no different than other Soar rules, potentially, they could be learned by Soar's symbolic learning mechanisms. While much more research is needed to make an agent that can learn to compose its own state representations via imagery and perception, the path for this research is much clearer than the alternative, where this composition must be handled in an isolated module in the perception system.

B3. Simulative imagery provides the agent with the ability to abstractly model actions that are non-deterministic at the abstract level.

This benefit is demonstrated by the ReLAI and RRT agents, all of which use imagery predictions as non-deterministic action models. That is, they utilize the knowledge that an imagery prediction is a prediction of a future state resulting from a particular action, rather than simply an inference about the current state. In particular, the comparison of ReLAI to imagery-augment direct state abstraction in the pedestal blocks world (Figure 13) clearly shows this difference, and demonstrates that action modeling provides a separate benefit beyond B1.

Simulative imagery is necessary for this benefit. These agents all take advantage of the non-deterministic nature of the model predictions, and non-deterministic prediction would not be possible in a system that only represented the problem at one level of abstraction. In extreme cases, agents use abstract state representations that are almost completely worthless for serving as the basis of an action model, and rely entirely on imagery for that capability. The backward-looking state predicates in the Fast Eddie agent are an example of this, and the RRT agent has a similar character. That agent uses an abstract state representation, but essentially the only properties encoded are whether or not certain configurations are reachable or not from other configurations. An abstract action model in terms of that information would not work well at all, since nothing in that state representation can be used to predict future state transitions.

The RRT agent also provides the clearest demonstration of why simulative imagery, rather than concrete routines alone, is necessary to fully achieve this benefit. Simulative imagery creates persistent changes in the concrete representation, and without that capability, imagery-based action modeling steps cannot be chained together. The RRT agent needs to model actions many steps in the future, so it needs the concrete results of previous action modeling operations—the car objects stored in the spatial scene during planning—to be persistent. If each car object was transient, and the changes to the scene disappeared once the agent inferred that a particular configuration was reachable, it would be impossible for that agent to create plans requiring intermediate configurations.

B4. Simulative imagery allows decisions to be made using an abstract representation while predictions are made using a concrete representation, allowing each process to use the representation that allows the most efficiency.

This benefit is demonstrated by all of the agents here that make predictions: the ReLAI and RRT agents. The benefit of abstraction for decision-making, if not apparent, is demonstrated in tasks like the pedestal blocks world. As shown in Figure 12, if a concrete representation is used for decision making in tasks like this, no learning is possible, since repeated concrete states are never encountered. This is an extreme example of the inefficiency of making decisions based on concrete information.

For making predictions, all of the ReLAI arcade game agents benefit from the fact that many aspects of the dynamics in those games can be easily approximated by linear translation of objects through space at a constant speed. With concrete information, it is simple to track and project forward this type of motion, but more abstract representations would lose this regularity. For example, consider using probabilistic abstract action models in those tasks. Leaving aside the issue of lost accuracy inherent in that approach (compared to precise imagery predictions), creating or learning abstract action models would be very difficult compared to what is necessary for the concrete models used here. The agent (or programmer) would have to observe many

abstract state transitions to determine, for example, the probability that the frog in Frogger II will collide with an obstacle in the next state if there is an obstacle above it in the current state and it issues an "up" action. In contrast, for a concrete action model, it is suitable simply to monitor the speeds of the objects in question and project them forward, an approach that requires very little learning or programming.

The RRT agent similarly demonstrates this. Since the motion in question is the result of a control process over concrete information, it is simple to predict the motion in terms of that same information.

B5. Concrete control allows continuous control processes to be used in conjunction with abstract symbolic reasoning, mitigating the irreducibility problem.

and **B6. Simulative imagery of concrete control allows symbolic reasoning over continuous processes, eliminating the need for symbolic characterization of controller performance, further mitigating the irreducibility problem.**

Both of these benefits were demonstrated by the RRT agent exclusively, since the other agents did not address precise control. As is discussed more fully in Section 6.4, the motion planning task here is fundamentally irreducible. Concrete controllers simplify the action space of the agent, and simulative imagery of concrete control allows the agent to use a controller that entails a complex interaction with the environment, as it locally steers toward a goal while being biased away from obstacles.

B7. Architectural representation conversion encapsulates complex, common processes, rather than requiring task-specific knowledge.

This benefit is demonstrated through the collection of agents presented here. The SVS architecture includes processes that are used by every agent. For example, every agent requires object collisions and distances to be detected, capabilities supported by the predicate extraction system of SVS. For arbitrarily-shaped convex polyhedrons, as SVS supports, these calculations entail solving fairly complex computational geometry problems. However, the predicate extraction system encapsulates the processes, and

each agent does not need task knowledge for how to calculate intersections or distances between objects.

For shared imagery operations, all of the arcade game ReLAI agents use the same basic linear-translation motion model, which does not need to be reprogrammed for each task. It does need parameters for each object, which can be either provided directly or learned as the agent observes the motion, but the processing beyond setting parameters is architectural and does not need to be provided by task knowledge. The car controller used by the RRT agent can be considered task knowledge; however, its operation is supported by architectural mechanisms in SVS that interface motion models to working memory and the spatial scene. The predicate projection system in SVS is also task-independent and architectural; however, it was demonstrated in only a single task here (the pedestal blocks world).

B8. Perception/action reuse provides for a parsimonious set of architectural processes.

The implementation of the architecture itself demonstrates this benefit. It would be possible to come up with a functionally identical system that does not use common architectural elements when dealing with the outside world and when dealing with imagery, but the implementation here indicates that this is unnecessary. The predicate extraction system does not consider whether the objects it operates over are real or imagined, allowing exactly the same high-level perception mechanisms to operate in real or imagined scenes. For instance, determining the state in the ReLAI and direct abstraction arcade game agents often involves the exact same predicate extraction processes in the architecture, but operating over perception in the direct abstraction case and imagery in the ReLAI case.

If the RRT agent were to be hooked up to an actual robot, it would similarly demonstrate reuse in the action system. As implemented, the agent simulated a controller that could (theoretically) be used for real actions, mimicking an approach

used by other systems (e.g., Leonard et al., 2008) which do reuse controllers for external action and imagery.

Taken as a whole, the set of theoretical aspects (A1-A8) leads to a set of benefits (B1-B8). Some of these aspects, via the benefits, lead to improvements in task performance, and some lead to improvements in generality. An architecture including all of the aspects has been implemented, and has been used to solve spatial problems while demonstrating each of the benefits.

8.2 Perceptual Abstraction, Irreducibility, and the Imagery Debate

A long-standing debate in psychology has been over the nature of mental imagery. To some, this is a debate over whether mental imagery is supported by propositional (symbolic) or depictive (picture-like) representations (Kosslyn et al., 2006). Others have posed the question as whether or not experimental data can disprove that "the process of imagistic reasoning involves the same mechanisms and the same forms of representation as are involved in general reasoning, though with different content or subject matter" (Pylyshyn, 2003), with the implication that those mechanisms are likely propositional.

This has been a difficult issue to resolve, since, in principle, both formats are able to represent the same information, and equivalent propositional and depictive accounts can be formulated to account for any behavioral data. However, other constraints can be taken into account, such as brain data, theoretical parsimony, or efficiency, to aid in identifying the underlying mechanisms (Anderson, 1978).

An abundance of brain data has been collected, largely supporting the hypothesis that imagery is a distinct process involving depictive representations (e.g., Kosslyn et al., 2006). Computational experiments have also examined efficiency characteristics of reasoning with different representational formats (e.g., Funt, 1980; Glasgow & Papadias, 1992; Huffman & Laird, 1992; Kurup & Chandrasekaran, 2006; Larkin & Simon, 1987; Lathrop, 2008; Shimojima, 1996; Tabachneck-Schijf et al., 1997). While not all of

these works directly addressed the imagery debate, all achieved results indicating that different representational formats afford different functional benefits, supporting the hypothesis of depictive imagery.

The architecture here also demonstrates an efficiency benefit of processing with different representations (B4). Beyond this, though, the examination of the perceptual abstraction and irreducibility problems can add to the imagery debate. As stated above, in principle, both abstract propositional and concrete depictive representations are able encode the same information. However, if the poverty conjecture is true, the proposal that an agent could behave intelligently using solely an abstract propositional representation becomes difficult to support.

If there exists no task-independent qualitative (abstract propositional) representation of space, an intelligent agent will need to encode different task-specific properties as new spatial tasks are encountered. This is what makes perceptual abstraction difficult. However, as demonstrated above, imagery within a concrete representation can mitigate this aspect of the perceptual abstraction problem (B2). This is then an argument supporting the hypothesis that imagery does not use an abstract propositional format, since the functional benefits of using imagery in this case derive from the fact that it is *not* abstract.[30] As stated by Forbus (1993), in reference to the MD/PV model discussed in Section 7.1,

> "If true, what does [the poverty conjecture] tell us about mental imagery? It suggests that there exists a set of commonplace tasks, such as understanding mechanical systems and reasoning about motion through space, that require representations that are richer than sparse propositional descriptions, whether performed by person or machine. Thus the question of whether or not imagery can be accounted for by sparse propositional representations comes down to whether or not the poverty conjecture is true."

[30] This argument does not directly support imagery using a depictive representation, only a concrete representation (one that encodes many details). Depictive representations are concrete, but more properties are needed for a representation to be depictive (see Kosslyn et al., 2006). Typically, depictive representations in a computer are array-based (e.g., a bitmap), where concrete representations, such as the spatial scene in SVS, may not be.

The irreducibility problem, along with its proposed solution in the form of concrete control and simulative imagery, similarly indicates a need for imagery in a concrete representation. In this case, in order to issue actions that are contingent on precise details of the environment, a concrete representation which captures all of those details is functionally useful. Strictly speaking, a concrete representation isn't *necessary* for this capability, as control processes can be reactive to details of perception without constructing a coherent representation (Brooks, 1991). However, as has been argued above, intelligent reasoning *about* control processes may not always be possible without the ability to simulate the results of those control processes in the particular situation within a concrete representation. Again, this indicates a functional benefit for imagery based in a concrete (and not abstract propositional) representation.

In both of these cases, intelligent reasoning in terms of abstract propositions is made possible only through the use of imagery in a concrete representation. The chief reason for the use of imagery is not that the imagery representation allows for more efficiency, but rather that the problem cannot be represented well in terms of abstract information alone. This is either because a task-independent architecture without imagery would not be able to make the relevant abstract distinctions, or because the problem is fundamentally irreducible to a form where it can be reasoned about in terms of abstract propositional information alone.

Essentially, creating a detailed, task-independent theory capable of addressing complex problems leads to functional arguments, beyond efficiency of processing in particular representational formats, that provide support for the hypothesis that imagery is not supported by an abstract propositional representation. While the analysis here supports the use of a concrete representation in general, rather than specifically a concrete depictive representation, given the evidence from brain imaging studies, depictive representation is a good hypothesis for how concrete representation might be manifested in the brain.

8.3 Soar/SVS as a Cognitive Model

Beyond the connection to the imagery debate, the work here could also be applied more directly to investigate cognitive modeling in spatial tasks. The Soar/SVS architecture has been presented as an AI system, but it could be adapted for use as a more precise cognitive model.

The architecture here does not include the timing, attention, capacity, imagery accuracy, or imagery maintenance constraints that would be needed for precise modeling. In addition, most of the agents presented here do not capture human-like reasoning in their tasks. A human playing an Atari game likely would be unable to imagine the consequences of each action four times per second, and a person would not use an algorithm like RRT to plan motions.

However, the purpose of these agents has been to provide straightforward demonstrations of the underlying functional benefits of the mechanisms of the architecture. When constraints are added, the approaches used in these demonstrations become infeasible, but the essential underlying benefits will remain. An agent using symbolic representations to make decisions encounters the perceptual abstraction and irreducibility problems regardless of what additional timing constraints, memory capacity constraints, etc., are present. If the architecture includes some version (however limited) of the aspects discussed here, the agent can likely get a functional benefit from using them.

For example, if the human cognitive architecture is roughly similar to a constrained version of what is presented here, it is reasonable to predict that a human attempting to achieve optimal behavior in the pedestal blocks world task would imagine the consequences of actions in order to avoid collisions. This is a prediction that arises from a few very basic premises:

(1) actions are chosen on the basis of some representation that can be approximated by abstract symbolic structures

(2) the human visual system cannot infer future block collisions in a bottom-up manner
(3) some representation that can be approximated by a concrete spatial representation is present
(4) imagery capability is present, and can be used to infer future block collisions to a useful degree of accuracy
(5) the subject is suitably motivated to perform well, and has a reasonable amount of time to choose each action

Similar predictions about the use of imagery in control tasks based on this work have been published elsewhere (Laird et al., 2010; Wintermute & Laird, 2009), however, to date, no experimental data has been collected. A line of research collecting human data in tasks like this could be useful not only to test the basic hypothesis that imagery will be used in problems where abstraction is difficult, but a rich collection of data could also constrain and help refine the details of the underlying architecture.

It should be noted that predictions here define "imagery" as it is defined in Chapter III, in terms of processing within a concrete representation. This contrasts with other definitions of imagery, such as imagery involving processing in a depictive representation (which is a specific form of concrete representation), or imagery as defined by subjective reports of "imagining" something. Imagery processing as defined here could be manifested in behavior independently of whether or not "imagining" is reported or brain activation indicating a depictive representation is present. For example, reaction time should be slower for action choices that require an intervening imagery operation than those that do not (which choices fall into which category being a prediction emerging from the details of the architecture). More experimental work is needed to determine the degree to which these definitions of imagery, and the data collected to support them, overlap.

8.4 Contributions

The main contributions of this thesis include the following:

The identification of meta-problems inherent in designing symbolic decision-making architectures

The veridical perception, perceptual abstraction, and irreducibility problems provide a useful decomposition of some of the issues behind creating a symbolic architecture capable of general intelligence. The problem of architecture design has not been decomposed this way in prior work.

The identification of imagery as a means to mitigate the perceptual abstraction problem

While previous systems have used imagery, its primary functional role has been as a more efficient means of inferring certain spatial properties compared to an equivalent symbolic representation. Here, instead, the use of imagery as a means to infer symbolic properties that the perception system of the agent would otherwise be unable to encode is demonstrated.

The identification of continuous control and imagery as means to mitigate the irreducibility problem

The existing technique of integrating symbolic reasoning with encapsulated continuous controllers is analyzed here with respect to the irreducibility problem. In addition, the existing technique of motion planning through control simulation is analyzed with respect to the irreducibility problem and the concept of simulative imagery.

In both of these cases, the analysis allows the prior work to be understood as means of mitigating the irreducibility problem in a general architecture. In addition, the analysis allows the potential for the principles behind the prior approaches to be generalized to apply outside of particular tasks.

A comprehensive, generally-stated, theory for integrating imagery and control in a symbolic cognitive architecture

Chapter III outlines the essential aspects of a theory incorporating imagery and continuous control into an architecture in such a way that perceptual abstraction and irreducibility can be mitigated.

This theory also serves to precisely define what is meant by "imagery". The definition of imagery here emphasizes the relationship between processing in terms of representations at different levels of abstraction as the defining characteristic of imagery. This way of defining imagery differs from previous accounts, which typically instead emphasize either human subjective experience, the type of content in the representation (e.g., visual or spatial information), or the "depictiveness" of the representation as defining characteristics of imagery.

An analysis of specific functional benefits afforded by the architectural theory

The theory has been mapped onto eight specific functional benefits. These benefits more precisely describe how the theory works to mitigate the perceptual abstraction and irreducibility problems, but also motivate why the particular integration scheme of the theory is appropriate.

An implemented, functional architecture embodying the general theory

Actually implementing the general theory requires a large number of details to be worked out that are left unspecified in the theory. The architecture presented here, Soar/SVS, outlines a candidate implementation of the theory that specifies all details. While the description here only outlines the broad construction of Soar/SVS, the fact that a running system exists demonstrates that all of those details can be (and have been) worked out.

The symbolic reasoning components of the implementation are directly inherited from the Soar architecture. The design of the SVS extension to Soar is in many ways a

refinement of Scott Lathrop's SVI system. However, SVS has important components that are completely novel, such as the motion processing system, and others that are substantial elaborations of SVI, such as the predicate projection system.

An integration of reinforcement learning and imagery, including theoretical analysis and a working implementation

The ReLAI algorithm represents a novel integration of reinforcement learning and imagery. Outside of architectural concerns, the algorithm is useful in cases where abstraction is difficult, but concrete, short-term predictions can be made. ReLAI has been theoretically analyzed in order to better understand the conditions under which it can result in convergence to the optimal policy. In addition to the algorithm itself, this work includes an analysis connecting the architectural principles presented earlier (e.g., perceptual abstraction) to related theoretical work describing Markov Decision Processes.

The ReLAI algorithm has also been implemented in the Soar/SVS architecture, and positive results are shown for its application in four different problems.

Demonstrations of imagery agents for complex problems, instantiated in a task-independent architecture and analyzed with reference to the functional benefits of the theory

The Soar/SVS architecture, using the ReLAI algorithm, has been applied to three different arcade game tasks. These agents demonstrate that simulative imagery is usable and useful in complex tasks that were not designed as imagery evaluation domains. More specifically, they demonstrate that the ReLAI algorithm works in those tasks, and even more specifically, they demonstrate that the Soar/SVS architecture is complete enough that agents for those tasks can be instantiated within it. These agents were also analyzed to provide case studies of the particular benefits of the underlying architectural theory.

A demonstration of sampling-based motion planning instantiated in a task-independent architecture, analyzed with reference to the functional benefits of the theory

Motion planning algorithms have typically been developed and used outside of cognitive architectures. The implementation of the RRT algorithm here maps some aspects of the algorithm onto Soar/SVS architectural processes, while some aspects are embodied as knowledge. This demonstrates that an agent instantiated in Soar/SVS can use this algorithm, or the principles behind it (simulation of control), in order to aid in planning its actions. Moreover, the agent provides a case study which demonstrates the benefits of the underlying architectural theory, particularly those benefits related to handling irreducibility.

An analysis of the implications of the work here as applied to cognitive modeling and the imagery debate

Psychological research on mental imagery served as a source of inspiration in this work, and, through the above contributions, led to advancements in the functionality of symbolic cognitive architectures. To reflect this progress back to psychology, the results here were analyzed to determine how they might inform future cognitive models and the imagery debate.

8.5 Future Work

This research indicates many directions for future work, some of which have been indicated as they have come up in prior sections. Here, a few of these directions will be discussed in detail.

Continuing the direction of prior work analyzing the processes of predicate projection and motion simulation in isolation (Wintermute & Laird, 2008, 2007), the memory retrieval and image composition process described in Section 4.5.2 could be further elaborated and enhanced. A technical report (Wintermute, 2009b) covers some of the open questions in this area. Particularly, the question of how inter-object

transformations are used and stored could use further study, along with the interaction of mental imagery and episodic memory (Nuxoll & Laird, 2007). This interaction should provide functional benefits, as the agent can compactly store the symbolic structure of the scene, and later recall it to derive further information not originally encoded. In addition, it will move Soar towards a comprehensive theory about the role of imagery and object prototypes (Posner & Keele, 1968) in memory.

Other means of interaction between spatial memory in SVS and symbolic memory and Soar also need to be investigated. Currently, the system has no good way of reasoning about spatial information that is not object-based. For example, places and gateways are not "objects", but can be usefully used to allow reasoning about empty space in the environment (Beeson et al., 2010). Similarly, an agent may find it useful to recognize locations in space that allow for interaction, such as the location on a door handle where it can be gripped to open the door (Klingbeil et al., 2008): roughly, these are similar to affordances (Greeno, 1994). It is currently unclear if this sort of entities (places, gateways, affordances) should be represented in the system differently than objects currently are, or if they could somehow emerge from the system without being explicitly represented. These entities seem to relate to properties of how multiple objects in space relate to each other, so it is possible they could be algorithmically calculated based on the objects in the scene.

Outside of the implementation in Soar/SVS, this work has introduced a broader theoretical architecture to account for the perceptual abstraction and irreducibility problems. An important (but difficult) direction for future work is to extend this theory to cover the remaining meta-problem, the veridical perceptual problem. This would involve proposing further architectural aspects[31] to support processes such as identifying what objects exist along with their spatial details, and instantiating that theory in an implemented architecture, such as a future version of SVS.

[31] Lathrop has proposed a rough sketch for these aspects as part of his work with SVI (Lathrop, 2008).

Progress is being made in robotic perception, enough that robots using internal spatial representations are becoming common in research environments. However, recognizing objects (beyond walls and obstacles) remains a challenge. Ideally, SVS would be able to internally retrieve the previously-learned complete 3d structure of an object based only on vision from one direction, but that remains at the fringe of (if not beyond) what is currently possible. However, it is possible that defining higher-level parts of the system can provide guidance on how low-level perception could be improved. For example, knowledge could play a strong role in disambiguating perception, or adjusting behavior to better handle inadequate perception. This is an interaction that may not be adequately present in systems that study perception in isolation from cognition. Studying this interaction is a focus of the ADAPT project (Benjamin et al., 2004) discussed in Section 7.2.

8.6 Conclusion

The goal of this thesis has been to investigate cognitive architectural structures to support intelligence in spatial tasks. This has led to a general-purpose architecture, extending Soar to support processing at multiple levels of abstraction through spatial imagery and continuous control. Theoretically, this work has addressed two fundamental issues in creating a general-purpose cognitive architecture: the perceptual abstraction and irreducibility problems. More practically, it has increased the breadth of problems Soar is able to address, and the performance it is able to achieve in those tasks. While it is difficult to predict the trajectory of future research, I hope that this work is a step on the path toward a solid computational understanding of human-level intelligent behavior.

Appendix: Quantities and Symbolic Representation

From the perspective of the development of the Soar architecture, a broad theme in this work has been the investigation of how Soar can handle problems where quantitative information is important. Soar has had long-standing support for quantitative numerical processing within working memory; however, this sort of processing has often been considered a programming convenience, rather than a theoretically important aspect of the architecture. In this appendix, the work here with imagery is used to inform a discussion of how the theory and implementation of quantitative processing in Soar might be progressed in future work. In addition, the current integration of quantitative processing informs a discussion of how the integration of SVS might be improved in the future.[32]

Soar supports the use of integers and floating-point numbers in its working memory. These quantities can be compared on the left hand side of rules, for example, a rule can be written to match only when some quantity A is greater than B. Soar can also mathematically manipulate quantities on the right hand side of rules, for example, a rule can be written to add a number C to working memory which is the result of A multiplied by B. This processing can be viewed very simply, as a useful but theoretically unfounded means for Soar agents to solve math problems, providing architectural processing that can replace the symbol manipulation. For example, a purely-symbolic model of multi-column subtraction (e.g., Rosenbloom et al., 1991) could be replaced by a single rule that calls the subtraction function on its right-hand side.

However, quantitative processing is also useful for tasks where the agent is not externally dealing with literal numbers. For example, an agent might use quantities in

[32] This appendix assumes some level of familiarity with the details of the Soar architecture beyond what is presented in this document.

working memory to encode distances to objects it can perceive, which are then compared to find the closest object to the agent. Taking a broader perspective, then, this processing can instead be viewed as a generic means of handling quantitative information, which is not necessarily connected with the external symbols (words and characters) people use to represent quantities.

It is informative to compare imagery processing in Soar/SVS to the existing quantitative processing in Soar. In SVS, quantitative information describing spatial properties, rather than being explicitly represented in symbolic working memory, is instead represented there by perceptual pointers. When perceptual pointers are processed by the symbolic aspects of Soar, they are treated as any other symbolic structure. However, symbolic processing can compose queries and imagery commands for SVS that include perceptual pointers. When a query is processed by SVS, the underlying quantitative information is accessed, the relevant property is calculated, and (typically) a qualitative symbolic result is created in working memory. When an imagery command is processed, similarly, the underlying quantitative information associated with the perceptual pointer(s) is accessed, a new object is added to the scene, and a perceptual pointer for that result is added to working memory, which can then serve as the basis for further processing.

With some caveats (outlined below), the quantitative numerical processing existing in Soar can be viewed in a similar light, as "mathematical imagery". In this conception, "number pointers" exist in working memory. These are arbitrary symbolic structures that are processed like any other such structure. Number pointers as implemented happen to have human-readable string representations like "**1.342**", but that is as arbitrary as **symbol3511** from the perspective of the symbolic processing system. Mathematical predicate extraction is possible, allowing symbolic structures to be created describing qualitative relationships between number pointers (e.g., **greater-than**, **less-than**, **equal-to**). Comparisons on the left-hand sides of rules support this. Mathematical imagery is possible, where specialized processing accesses the underlying representations associated with a set of number pointers, and creates a new

number pointer in working memory associated with the result of the specified "imagery" operation. This capability is supported via the mathematical right-hand side functions in Soar. Mathematical imagery operations supported by Soar then include addition, subtraction, multiplication, and division.

From this perspective, there are four differences between the implementation of mathematical imagery and spatial imagery.

(1) The underlying quantitative information differs. Mathematical imagery involves single numbers, where spatial imagery primarily involves polyhedrons in 3d space.

(2) The integration of quantitative processing with Soar's decision cycle differs. Mathematical predicate extraction is implicitly invoked as rules are matched, such as if a rule includes a less-than condition, and mathematical imagery is captured through right-hand side functions. Spatial predicate extraction and imagery are instead mediated through working memory: rules set up declarative predicate extraction query or imagery command structures, and the results are added to working memory by SVS.

(3) The integration of quantitative content with Soar's symbolic memories differs. For mathematical imagery, these memories store the actual underlying quantities, rather than simply number pointers. If the agent computes that the result of multiplying 3.2 times 8.1 is 25.92, working memory will store that exact value (meaning rules could match against it), and a chunk, semantic memory, or episodic memory created for this event will store the exact quantities involved. In the spatial imagery case, though, if the agent imagines the convex hull of two shapes, resulting in a new shape, the symbolic memories will only encode the string values of the perceptual pointers, not the underlying quantitative structures involved.

The storage of complete quantitative content in working memory also results in an imperfection in the analogy of quantities in working memory as similar to perceptual pointers. A perceptual pointer can track an object whose quantitative details change over time (e.g., a moving object), where the identity of the "number pointer" would

necessarily change as the underlying quantity changes. However, this is not a great difficulty, as it is easily remediated through a level of symbolic indirection. For example, the following structure might be used in working memory:

```
(number N1)
(N1 ^id addend ^value 12.5)
```

In this case, the **id** augmentation, rather than the **value** itself, fulfills a role similar to an SVS perceptual pointer. If the **value** is later changed, it can still be accessed via the associated **id**, which stays constant.

(4) Quantities resulting from mathematical imagery can be directly output (reported) by the agent, while it would be much more difficult to create an agent that can report the quantitative details of objects resulting from spatial imagery. For example, it is trivial to make an agent that uses right-hand side functions to calculate, for example, that 1.23 times 51.67 is 63.551, and adds a string to its output asserting so. There is no straightforward equivalent to this capability for spatial imagery.

This analysis leads to five questions that could be addressed in further work:

Should imagery be integrated at the rule level, rather than mediated through working memory?

The integration of quantitative numerical processing at the rule level works well, and results in simpler agent development, as knowledge is not needed to set up queries and to parse responses. Spatial imagery could be similarly set up in this way, where predicate extractions are evaluated as part of matching the left-hand side of a rule, and imagery operations are set up as right-hand side functions that return a perceptual pointer to the result.

However, predicate extraction and imagery operations can take much more time to execute than it takes to compare numbers or perform basic arithmetic. This integration could then substantially slow down the rule-matching phases of Soar's decision cycle. At a higher level, though, the total number of operations the agent needs to perform may

not necessarily change, but might simply be moved into a different part of the decision cycle. Much like it is possible to create symbolic rules that are difficult for Soar to match, it is possible to set up predicate extraction operations that take a long time to process. This would remain true whether those operations were invoked by a query built in working memory or were directly encoded in the left-hand side of a rule.

Future development of SVS may change this, though. The current memory-mediated integration of SVS could easily be adapted for asynchronous processing, where complex operations would occur over multiple decisions, allowing the agent to remain reactive during those operations. Rule-level integration could not as easily support asynchronous processing.

In addition, a flaw in the scheme used for quantitative numerical processing in Soar is that rules involving comparisons and mathematical functions cannot be learned by the architecture. Soar's chunking mechanism, which learns new rules, cannot create rules with these features. In contrast, rules used to build declarative command and query structures are (in principle) learnable by the architecture. In that way, the SVS interface scheme is more in line with that aspect of the Soar theory, indicating that it should be retained.

How should imagery structures be integrated with symbolic structures?

In principle, there is no reason the architecture could not be modified to store quantitative spatial structures with its symbolic memories, as is currently done with numbers. However, this does not seem to be a worthwhile approach. To completely mimic the numeric scheme, every detail of every object (including its shape and position in space) in the spatial scene would have to be stored every time an episodic memory is stored. In addition, if a rule was learned through chunking, it might test one or more complete spatial structures in its conditions and produce one in its actions.

Since there are so many details in the spatial representation, it would be unlikely that any real object would reappear with the same quantitative details. This means that the

amount of information stored in episodic memory would grow much faster than it does now. In addition, chunks would be created that match against complete quantitative structures. These chunks would only very rarely match (outside of situations where an episodic memory has been retrieved of the situation that resulted in the chunk).

SVS currently includes a perceptual LTM independent of Soar's long-term symbolic memories. This LTM currently has no theory for learning, but does store complete quantitative structures. These structures are considered to be prototypes representative of object classes, though, rather than instances. Objects in the spatial scene are labeled with perceptual pointers to associated long-term memory prototypes. While much further work is needed to fully integrate SVS with symbolic learning in Soar, a proposed scheme is as follows:

- Perceptual LTM should be considered an extension of Soar's symbolic semantic memory, but augmented with spatial details
- Soar's episodic memory should capture perceptual pointers only, and not spatial structures
- Agents should deliberately recall spatial aspects of episodic memories by using imagery to compose a scene, retrieving its components as prototypes from perceptual LTM
- Imagery processing should not be reduced to rules via chunking

In this scheme, only limited learning over spatial information is possible, but it has the advantage of requiring much less storage space than alternatives.

Should quantitative numerical processing gain theoretical status?

As was mentioned, quantitative processing in Soar is often regarded as a programming convenience rather than an important part of the theory. A useful research direction to consider is to bring quantitative processing into the Soar theory, possibly substantially changing the implementation along the way. Newell (1988, 1990) argued that a "basic quantitative code" is necessary to represent quantities in working memory to account

for phenomena such as perceptual judgments (which are often modeled as evidence accumulating to meet a threshold), and since perceptual and motor interfaces likely deal with quantitative information:

> "It is easy to show that an architectural mechanism is required [for the basic quantitative code], since the issue is the transduction from perceptual signals, which are clearly quantitative (intensity, direction), to symbols in Soar's representation, and from such symbols to motor signals, which again are clearly quantitative (force, direction)." (Newell, 1990, p. 437)

However, these particular needs might be superseded with SVS, since the quantitative connections between perception and action can be mediated through SVS, rather than working memory, and since the predicate extraction system might internally handle accumulation of evidence, only adding the result to working memory once a threshold is reached.

Regardless of this, the use of quantities in working memory certainly seems to provide a functional benefit in some cases, and a detailed investigation into the subject would be valuable. One possible avenue for this is to expand on the analogy above, and consider quantitative numerical processing as another form of imagery. Others have suggested that amodal imagery operating on individual quantities might be a useful addition to Soar (Hines, 2010). This research could lead to, for example, an investigation of what operations lead to improved performance and generality, as this thesis presents for spatial imagery. If this approach were taken, it would make sense to consider handling mathematical and spatial imagery in similar ways. That is, the answers to the above questions might also apply to mathematical imagery in addition to spatial imagery.

Should quantitative processing in Soar be eliminated in favor of SVS?

Finally, as mentioned above, much of the value of mathematical processing in Soar has been superseded by SVS processing. Previous agents utilized this processing for capabilities now supported by SVS, for example, determining distances between points

in space. In addition, other mathematical problems could likely be solved via analogies to spatial problems. As mentioned in Section 4.4, though, even when SVS is used, it is still useful to represent some information quantitatively in working memory so that comparisons can later be made (in that case, between distances). However, a minor change to the architecture could provide equivalent functionality in SVS without the need for quantities in working memory. Therefore, a detailed examination of whether some or all of this processing should be eliminated as redundant would be valuable.

References

Agre, P. E., & Chapman, D. (1987). Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*.

Anderson, J. R. (1978). Arguments concerning representations for mental imagery. *Psychological Review*, *85*(4), 249–277.

Anderson, J. R. (2005). Human Symbol Manipulation Within an Integrated Cognitive Architecture. *Cognitive Science*, *29*(3), 313-341.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*(4), 1036-1060.

Banerjee, B., & Chandrasekaran, B. (2007). Representations and Strategies for Solving Spatial Problems with Diagrams. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing* (pp. 183-188).

Barkowsky, T. (2002). *Mental Representation and Processing of Geographic Knowledge: A Computational Approach*. Lecture Notes in Artificial Intelligence (Vol. 2541). Springer.

Barsalou, L. W. (1999). Perceptual symbol systems. *Behavioral and Brain Sciences*, *22*(04), 577-660.

Barsalou, L. W. (2008). Grounded cognition. *Annual Review of Psychology*, *59*, 617-645.

Beeson, P., Modayil, J., & Kuipers, B. (2010). Factoring the mapping problem: Mobile robot map-building in the Hybrid Spatial Semantic Hierarchy. *International Journal of Robotics Research*, *29*(4), 428-459.

Benjamin, D. P., Lyons, D., & Lonsdale, D. (2004). ADAPT: A Cognitive Architecture for Robotics. In *Proceedings of ICCM-2004*.

Benjamin, D. P., Lyons, D., & Lonsdale, D. (2006). Embodying a cognitive model in a mobile robot. In *Proceedings of SPIE* (Vol. 6384, p. 638407). Presented at Intelligent Robots and Computer Vision XXIV: Algorithms, Techniques, and Active Vision, Boston, MA.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of*

Robotics and Automation, *2*(1), 14-23.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence*, *47*, 139-159.

Cassimatis, N. L., Trafton, J. G., Bugajska, M. D., & Schultz, A. C. (2004). Integrating cognition, perception and action through mental simulation in robots. *Robotics and Autonomous Systems*, *49*(1-2), 13-23.

Chandrasekaran, B. (1997). Diagrammatic representation and reasoning: some distinctions. In *Proceedings of the AAAI Fall Symposium on Diagrammatic Reasoning*.

Chandrasekaran, B. (2006). Multimodal Cognitive Architecture: Making Perception More Central to Intelligent Behavior. *Proceedings of the AAAI National Conference on Artificial Intelligence*, 1508-1512.

Chandrasekaran, B., & Kurup, U. (2007). A Bimodal Cognitive Architecture: Explorations in Architectural Explanation of Spatial Reasoning. In *Proceedings of the AAAI Spring Symposium on Control Mechanisms for Spatial Knowledge Processing in Cognitive / Intelligent Systems*.

Chandrasekaran, B., Kurup, U., Banerjee, B., Josephson, J. R., & Winkler, R. (2004). An Architecture for Problem Solving with Diagrams. In *Diagrammatic Reasoning and Inference*, Lecture Notes in Artificial Intelligence (Vol. 2980, pp. 151-165). Berlin: Springer-Verlag.

Cohn, A. G., Bennett, B., Gooday, J., & Gotts, N. M. (1997). Qualitative Spatial Representation and Reasoning with the Region Connection Calculus. *GeoInformatica*, *1*(3), 275-316.

Diuk, C., Cohen, A., & Littman, M. L. (2008). An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning* (pp. 240-247). New York: ACM.

Fajen, B. R., & Warren, W. H. (2003). Behavioral dynamics of steering, obstacle avoidance, and route selection. *Journal of Experimental Psychology: Human Perception and Performance*, *29*(2), 343-362.

Forbus, K. D. (1983). Qualitative reasoning about space and motion. In *Mental Models* (pp. 53-73). Hillsdale, New Jersey: Lawrence Erlbaum.

Forbus, K. D. (1993). Image and substance. *Computational Intelligence*, *9*(4), 377-378.

Forbus, K. D., Nielsen, P., & Faltings, B. (1991). Qualitative spatial reasoning: the CLOCK project. *Artificial Intelligence*, *51*(1-3), 417-471.

Funt, B. V. (1980). Problem-solving with diagrammatic representations. *Artificial Intelligence*, *13*, 201–230.

Gelernter, H. (1963). Realization of a Geometry-Theorem Proving Machine. In *Computers and Thought* (pp. 134-152). Cambridge, MA: MIT Press.

Givan, R., Dean, T., & Greig, M. (2003). Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, *147*(1), 163–224.

Glasgow, J., & Papadias, D. (1992). Computational imagery. *Cognitive Science*, *16*(3), 355-394.

Glasgow, J. (1995). A formalism for model-based spatial planning. In *Spatial Information Theory: A Theoretical Basis for GIS* (pp. 501-518).

Goetschalckx, R. U. (2009). *On the Use of Domain Knowledge in Reinforcement Learning* (PhD Thesis). Leuven, Belgium: Katholieke Universiteit Leuven.

Greeno, J. G. (1994). Gibson's Affordances. *Psychological Review*, *101*(2).

Grush, R. (2004). The emulation theory of representation: Motor control, imagery, and perception. *Behavioral and Brain Sciences*, *27*(03), 377-396.

Gunzelmann, G., & Lyon, D. R. (2006). Mechanisms for Human Spatial Competence. In *Proceedings of Spatial Cognition V*.

Harrison, A. M., & Schunn, C. D. (2003). ACT-R/S: Look Ma, no "cognitive-map"! In *Proceedings of the Fifth International Conference on Cognitive Modeling* (pp. 129-134).

Hernández, D. (1994). *Qualitative Representation of Spatial Knowledge*. Lecture Notes in Artificial Intelligence (Vol. 804). Springer-Verlag.

Hines, J. (2010). ODE imagery?. Message to the soar-group mailing list. Retrieved from http://sourceforge.net/mailarchive/message.php?msg_name=AANLkTilul7KhCux-ga_gBmEEcIN4upc0ETV90Sm-DUlR%40mail.gmail.com

Huffman, S., & Laird, J. E. (1992). Using Concrete, Perceptually-Based Representations to Avoid the Frame Problem. In *Proceedings of the AAAI Spring Symposium on Reasoning with Diagrammatic Representations*.

Kieras, D. E., & Meyer, D. E. (1997). An Overview of the EPIC Architecture for Cognition and Performance with Application to Human-Computer Interaction. *Human-Computer Interaction, 12*, 391-438.

Klingbeil, E., Saxena, A., & Ng, A. Y. (2008). Learning to Open New Doors. In *Proceedings of the AAAI 17th Annual Robot Workshop and Exhibition*.

Kosslyn, S. M., Thompson, W., & Ganis, G. (2006). *The Case for Mental Imagery*. New York: Oxford University Press.

Kuipers, B. (2000). The Spatial Semantic Hierarchy. *Artificial Intelligence, 119*(1-2), 191-233.

Kurup, U., & Chandrasekaran, B. (2006). Multi-modal Cognitive Architectures: A Partial Solution to the Frame Problem. In *Proceedings of The 28th Annual Conference of the Cognitive Science Society*.

Kurup, U., & Chandrasekaran, B. (2007). Modeling Memories of Large-scale Space Using a Bimodal Cognitive Architecture. In *Proceedings of the Eighth International Conference on Cognitive Modeling* (pp. 267-272).

Laird, J. E. (2008). Extending the Soar Cognitive Architecture. In *Proceedings of the First Conference on Artificial General Intelligence.*

Laird, J. E., Xu, J., & Wintermute, S. (2010). Using Diverse Cognitive Mechanisms for Action Modeling. In *Proceedings of ICCM-2010.*

Laird, J. E., Yager, E. S., Hucka, M., & Tuck, C. M. (1991). Robo-Soar: An integration of external interaction, planning, and learning using Soar. *Robotics and Autonomous Systems, 8*(1-2), 113-129.

Larkin, J. H., & Simon, H. A. (1987). Why a Diagram is (Sometimes) Worth Ten Thousand Words. *Cognitive Science, 11*(1), 65-100.

Lathrop, S. D. (2008). *Extending Cognitive Architectures with Spatial and Visual Imagery Mechanisms* (PhD Thesis). University of Michigan.

Lathrop, S. D., & Laird, J. E. (2007). Towards Incorporating Visual Imagery into a Cognitive Architecture. In *Proceedings of the Eighth International Conference on Cognitive Modeling*.

Lathrop, S. D., & Laird, J. E. (2009). Extending Cognitive Architectures with Mental Imagery. In *Proceedings of the Second Conference on Artificial General*

Intelligence.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

LaValle, S. M., & Kuffner Jr, J. J. (2001). Randomized Kinodynamic Planning. *The International Journal of Robotics Research*, *20*(5), 378.

Lehman, J. F., Laird, J., & Rosenbloom, P. (2006). A Gentle Introduction to Soar: 2006 Update. Retrieved from http://ai.eecs.umich.edu/soar/sitemaker/docs/misc/GentleIntroduction-2006.pdf

Leonard, J., How, J., Teller, S., Berger, M., Campbell, S., Fiore, G., Fletcher, L., et al. (2008). A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, *25*(10).

Li, L., Walsh, T. J., & Littman, M. L. (2006). Towards a unified theory of state abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics* (pp. 531–539).

Lindemann, S. R., & LaValle, S. M. (2003). Current issues in sampling-based motion planning. In *Proceedings of the International Symposium of Robotics Research*. Springer.

Lozano-Pérez, T., & Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, *22*(10), 560-570.

Madhavan, R., Messina, E., & Albus, D. J. (2006). *Intelligent Vehicle Systems: A 4D/RCS Approach*. New York: Nova Science Publishers.

Nason, S., & Laird, J. E. (2005). Soar-RL: integrating reinforcement learning with Soar. *Cognitive Systems Research*, *6*(1), 51-59.

Newell, A. (1988). The Basic Quantitative Code: Statement of the Problem. Retrieved from http://shelf1.library.cmu.edu/cgi-bin/tiff2pdf/newell/box00009/fld00552/bdl0001/doc0001/newell.pdf

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA, USA: Harvard University Press.

Nuxoll, A., & Laird, J. E. (2007). Extending cognitive architecture with episodic memory. *Proceedings of the AAAI National Conference on Artificial Intelligence*, *1001*.

Pinker, S. (1984). Visual cognition: an introduction. *Cognition*, *18*(1-3).

Posner, M. I., & Keele, S. W. (1968). On the genesis of abstract ideas. *Journal of Experimental Psychology*, *77*(3), 353-363.

Pylyshyn, Z. W. (2003). Mental imagery: In search of a theory. *Behavioral and Brain Sciences*, *25*(02), 157-182.

Pylyshyn, Z. W. (2001). Visual indexes, preconceptual objects, and situated vision. *Cognition*, *80*(1-2), 127-158.

Ravindran, B., & Barto, A. G. (2002). Model minimization in hierarchical reinforcement learning. In *Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation* (pp. 196–211).

Rosenbloom, P. S., Laird, J. E., Newell, A., & McCarl, R. (1991). A preliminary analysis of the Soar architecture as a basis for general intelligence. *Artificial Intelligence*, *47*(1-3), 289-325.

Shimojima, A. (1996). *On the efficacy of representation* (PhD Thesis). Indiana University.

Stein, L. A. (1994). Imagination and situated cognition. *Journal of Experimental and Theoretical Artificial Intelligence*, *6*(4), 393-407.

Stober, J., & Kuipers, B. (2008). From pixels to policies: A bootstrapping agent. In *Proceedings of the 7th IEEE International Conference on Development and Learning* (pp. 103-108).

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, *8*, 1038-1044.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Tabachneck-Schijf, H. J. M., Leonardo, A. M., & Simon, H. A. (1997). CaMeRa: A computational model of multiple representations. *Cognitive Science*, *21*(3), 305-350.

Tenenbaum, J. B., Griffiths, T. L., & Kemp, C. (2006). Theory-based Bayesian models of inductive learning and reasoning. *Trends in Cognitive Sciences*, *10*(7), 309–318.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, *38*(3), 58-68.

Ullman, S. (1984). Visual routines. *Cognition*, *18*(1-3), 97.

Wang, Y., & Laird, J. E. (2010). Efficient Value Function Approximation with Unsupervised Hierarchical Categorization for a Reinforcement Learning Agent. In *Proceedings of the 2010 International Conference on Intelligent Agent Technology*.

Wintermute, S. (2009a). Integrating Reasoning and Action through Simulation. In *Proceedings of the Second Conference on Artificial General Intelligence*.

Wintermute, S. (2009b). *An Overview of Spatial Processing in Soar/SVS* (Technical Report No. CCA-TR-2009-01). University of Michigan Center for Cognitive Architecture.

Wintermute, S. (2010). Using Imagery to Simplify Perceptual Abstraction in Reinforcement Learning Agents. In *Proceedings of the the Twenty-Fourth AAAI Conference on Artificial Intelligence.*

Wintermute, S., & Laird, J. E. (2007). Predicate Projection in a Bimodal Spatial Reasoning System. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence.*

Wintermute, S., & Laird, J. E. (2008). Bimodal Spatial Reasoning with Continuous Motion. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence.*

Wintermute, S., & Laird, J. E. (2009). Imagery as Compensation for an Imperfect Abstract Problem Representation. In *Proceedings of the 31st Annual Conference of the Cognitive Science Society*.