Imagery in Cognitive Architecture: Representation and Control at Multiple Levels of Abstraction

Samuel Wintermute

Abstract

Visual and spatial mental imagery processes seem to play a prominent role in human cognition, and AI systems have occasionally incorporated imagery-like processes in order to leverage functional benefits. Typically, these benefits have been characterized as the ability to perform more efficient inference through the use of specialized representation. However, as explored in this article, when considering the design of a cognitive architecture—a specification of fixed mechanisms underlying intelligent behavior—the functional benefits of imagery go beyond increased inference efficiency.

In a cognitive architecture, as in most AI systems, intelligent behavior is often contingent upon the use of an appropriate abstract representation of the task. When designing a general-purpose cognitive architecture, two basic challenges related to abstraction arise. The perceptual abstraction problem results from the difficulty of creating a single perception system able to induce appropriate abstract representations in any task the agent might encounter, and the irreducibility problem arises because some tasks are resistant to being abstracted at all. Key benefits of imagery relate to addressing these challenges.

As it is defined here, to support imagery, a concrete (highly detailed) representation of the spatial state of the problem is maintained as an intermediate between the external world and an abstract representation. Actions can be simulated (imagined) in terms of this concrete representation, and the agent can derive abstract information by applying perceptual processes to the resulting concrete state. Imagery works to mitigate the perceptual abstraction problem by allowing a given perception system to work in more tasks, since perception can be dynamically combined with imagery, and works to mitigate the irreducibility problem by allowing internal simulation of low-level control processes.

To demonstrate these benefits, an implementation is described, which is an extension of the Soar cognitive architecture. An agent in this architecture that uses reinforcement learning and imagery to play an arcade game and an agent that performs sampling-based motion planning for a car-like vehicle are described. The performance of these agents is discussed in the context of the underlying imagery theory.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
19 MAR 2011	**Journal Article**	**19-03-2011 to 19-03-2011**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
IMAGERY IN COGNITIVE ARCHITECTURE: REPRESENTATION AND CONTROL AT MULTIPLE LEVELS OF ABSTRACTION	**W56h2V-04-2-0001**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) **Samuel Wintermute**	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Automotive Research Center,University of Michigan,1231 Beal Ave,Ann Arrbor,MI,48109-2133**	8. PERFORMING ORGANIZATION REPORT NUMBER **; #21799**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) **U.S. Army TARDEC, 6501 E.11 Mile Rd, Warren, MI, 48397-5000**	10. SPONSOR/MONITOR'S ACRONYM(S) **TARDEC**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) **#21799**

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
N/A

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**51**	

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. The broad goal of this research is the development of a cognitive architecture to support behavior of this sort. A cognitive architecture is a theory of the fixed processes underlying intelligent behavior (Langley et al., 2009). Accordingly, hypotheses about architectural structures supporting intelligence in complex spatially-oriented tasks are necessary.

An important dimension through which these issues can be viewed is that of *abstraction*. In complex spatial tasks, as in most tasks, an agent can benefit from using an abstract internal representation of the structure of the task. Abstraction removes unnecessary detail, and is a key means by which learning and knowledge representation can be made tractable. However, the need for abstraction seems to conflict with the need to account for the rich detail of the spatial world, as is necessary for intelligence in tasks where those details matter. Since abstraction is a process supported by the perception and action systems of a cognitive architecture, it is a critical issue to consider when designing an architecture.

The basic conflict between abstraction and precision is analyzed in this article. As will be explained in the next section, there are two problems inherent in designing a general-purpose cognitive architecture capable of abstract representation while still maintaining the spatial precision necessary for intelligent behavior. 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.

Aspects of these challenges have manifested (and have been addressed) in research in several subfields of AI, including robotic motion planning, qualitative reasoning, and reinforcement learning. However, the root perceptual abstraction and irreducibility problems have not previously been identified and studied in a general manner outside of particular task domains. The development of a task-independent cognitive architecture presents a context where this more basic analysis is necessary.

In this article, the perceptual abstraction and irreducibility problems are introduced, and a theory of basic architectural mechanisms that can work to mitigate these problems is

presented. 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 imagery capability, where the concrete representation is internally manipulated, with the results feeding back to the abstract representation. 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 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 article 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; Kurup & Chandrasekaran, 2006; Ullman, 1984).

Imagery capability for visual and/or spatial information has been proposed as an important cognitive architectural component (e.g., Chandrasekaran, 2006; Lathrop, 2008). Much of the motivation for this inclusion has been drawn from psychological research. Arguments about the utility of imagery outside of psychological concerns—functional arguments for imagery—have also been made (e.g., Lathrop et al., 2010). Typically these arguments are based on research examining benefits in terms of increased inference efficiency afforded by imagery-like processing (Glasgow & Papadias, 1992; Huffman & Laird, 1992; Larkin & Simon, 1987), or through demonstrations of particular domains where imagery use is beneficial, such as planning to coordinate a team of military scout robots (Lathrop & Laird, 2009). The use of imagery presented here, as a means of mitigating the perceptual abstraction and irreducibility problems, shows different functional benefits of imagery beyond inference efficiency. In that way, this work can contribute towards a more thorough understanding of the role of imagery in cognitive architecture.

To elaborate on this, prior functionality-based examinations of imagery have assumed that, since abstract propositional representations and concrete perceptual representations can in principle encode the same information (Anderson, 1978), the primary functional role for imagery is to allow more efficient inference. However, the analysis here reveals that, due to the difficulty of solving the perceptual abstraction problem, 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. Furthermore, the irreducibility problem creates in an additional role for

imagery processing, as imagery allows low-level control processes to be internally simulated. This important connection with action control is missing in prior functional analyses of imagery.

These concepts lead to a difference in the broad way perceptual-level representation and imagery are understood in the context of a cognitive system. Rather than viewing imagery primarily as a more efficient means for addressing particular tasks, like solving geometry problems, or as a means to model human imagination, here, it is an integral part of the basic process of capturing the right details of the state of the outside world and the way that that state interacts with the agent's low-level control processes

That is, 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 this might also allow an agent to perform mental rotation, solve geometry problems, recall memories or construct imaginary scenes, but the tasks examined here are more basic, involving the immediate interaction between the agent and its environment.

In order to make traction in this analysis, for the tasks studied in this article it is assumed that a concrete representation encoding spatial properties is available. 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.

Section 1 -Motivation

1.1 Basic Assumptions

The goal of this work is to explore basic issues in architecture design. The space of all theories upon which an architecture might be based is huge, and not all possibilities can be addressed here. Therefore, here, discrete decision-making is assumed. That is, we assume an 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. For this work, the relevant 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 values of actions) cannot be a continuous function of the agent's internal state.

As will be explained in detail shortly, there are two basic implications of this assumption: 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 (Sutton & Barto, 1998) 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 and there is no discrete decision among actions (e.g., a feedback rule or neural network).

However, these assumptions do not mean that actions must be a function of symbolic perceptions alone. Previous perceptions and arbitrary background knowledge can be stored and influence decisions. In addition, non-symbolic processes can operate over symbolic information and effect decisions. For example, reinforcement learning adjusting control biases, 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 output—must be the direct result of a decision. 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, while other methods such as feedback control can be used at lower levels. Despite additional complexity in these cases, the two basic implications remain: symbolic information must distinguish between situations where decisions must differ, but as few situations as possible should be distinguished.

1.2 Motivating Tasks

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

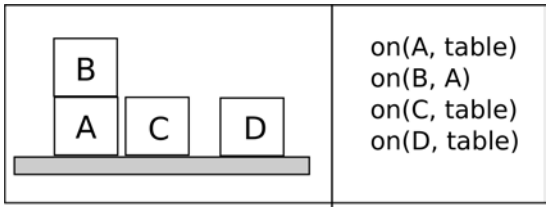

Figure 1: A simple blocks world task.

1.2.1 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**.

A straightforward way of addressing this task in a symbolic agent is to use a planning language such as that used in the STRIPS system (Fikes & Nilsson, 1971). The state is described in terms of abstract predicates[1], as shown on the right of the figure, and rules encode the consequences of actions in terms of those predicates. 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 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.

In the blocks world, for both of these agents, the **on** predicates (as shown in the figure) 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 problems, and must solve each one. In realistic environments, any given instance of the task might vary in its details: the blocks might be in slightly different positions, they might be different colors, or be slightly larger or smaller. However, the representation 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, details can be ignored so that many underlying problem instances are mapped to a single internal task representation. Any

[1] Throughout this work, predicate representations will be used as notational shorthand for generic symbolic representations. The use of predicates is merely to illustrate what data is encoded in terms of symbols, it does not imply that logical inference is used.

[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 5.4.

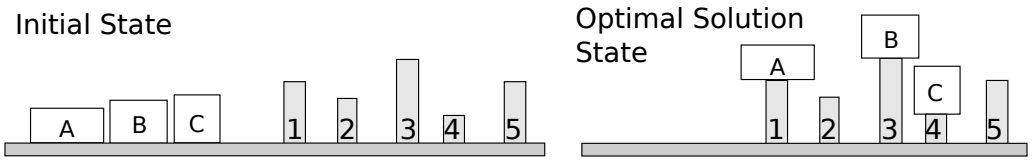

Figure 2: A pedestal blocks world problem and its optimal solution state.

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 these abstract agents, 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 sort of detailed representation *concrete*). If these coordinates are treated symbolically, any variation in the blocks, however minor, will cause the agent to perceive a completely different state. An agent using such a representation has very little generalization ability: it is extremely unlikely that two blocks in different problem instances would ever appear in the same precise position, so a reinforcement learning agent would need to learn a unique policy for each instance, and a planning agent would need unique rules for each instance.

In short, an agent with the ability to induce **on** predicates is able capture all of the right details of the blocks world to make correct decisions, and is able to do so while minimizing the number of distinctions made. This representation forms the ideal case for symbolic decision-making, as it is both compact and accurate.

1.2.2 The Pedestal Blocks World

To illustrate a situation where symbolic representation is not so simple, 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 first moves block **A** to some pedestal, 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 were necessary in the unmodified blocks world (**on** 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. 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 encodes a complex, 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. While one might propose that a generic perception system could provide **on** information for objects it sees, supporting blocks world tasks, the symbolic information necessary in this task is not such a simple, local, task-independent property of the environment. This raises the issue, then, of how a generic architecture could support solving pedestal blocks world problems.

1.2.3 Motion Planning for a Nonholonomic Car

In some cases, creating symbolic representations is even more difficult, due to the need for precise control.

Even in the blocks world, if a real robot is used, precise control is necessary. In a real robot, the final output of the agent is a set of motor voltages. Since real blocks can vary in size and shape, the actual outputs 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 like the blocks world, this aspect can be ignored: low-level controllers that continuously transform the details of the perceived state to output voltages can be incorporated in the system. 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).

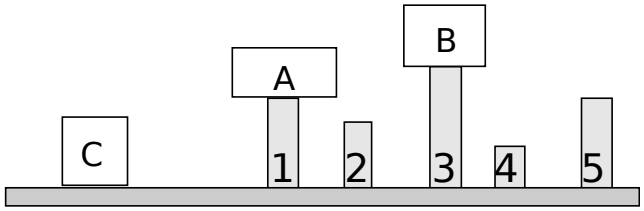

Figure 3: A situation in pedestal blocks world where nonlocal interactions are important if the agent is considering moving block C to pedestal 2.

For this encapsulated controller approach to work, though, to make intelligent decisions, the controllers 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. In the blocks world, if a robotic gripper is controlled to move physical blocks, the previously described approaches to the problem will work well if the gripper controller can reliably transition between states described by **on** predicates.

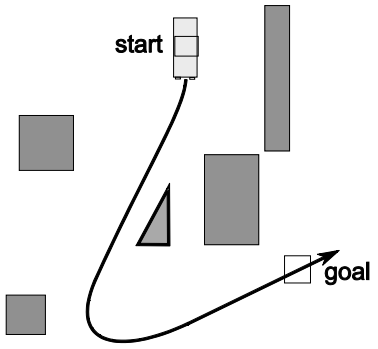

Figure 4: A nonholonomic car motion planning problem.

However, again, some tasks are not as straightforward. Motion planning, as it will be 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. 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 the 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 classes of robots, such as polygonal robots that can move in any direction, and for more complicated robots when tight maneuvering is less important.

However, in other situations, the encapsulated controller approach does not work. 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. 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). 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 another challenge to symbolic agents, which will be addressed shortly.

1.3 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 as a prerequisite for solving external problems.

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

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

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 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 decision should be made versus another. If two situations cannot be distinguished in terms of the available symbolic information, the agent will make the same decision in both. In addition, these symbols should distinguish as *few* states as possible—they should be abstract. If, instead of using **on** predicates, a blocks world agent encoded every detail of the problem in its symbolic representation (a concrete representation), planning or learning would be infeasible.

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

Perceptual Abstraction Problem: An agent must have the 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 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.

The word “irreducibility” here makes the most sense when the task is viewed as an MDP (as will be elaborated in Section 3). The size of an MDP may be “reduced” by identifying

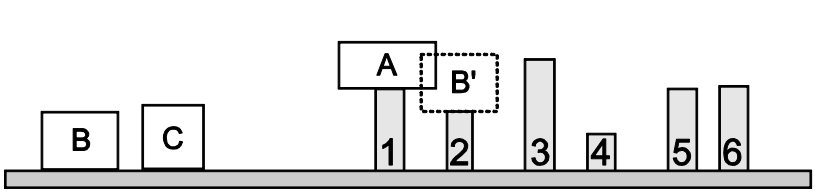

Figure 5: Imagery in pedestal blocks world.
The agent has imagined block b on pedestal2 (creating an imagined copy, B') and inferred the abstract predicates at right.

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 problem can occur in the context of any symbolic representation scheme, not just MDPs.

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 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 transformation 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.

An 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 intelligent behavior 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 that must be addressed by an architecture capable of supporting intelligent behavior in the same breadth of tasks as humans.

1.4 Imagery for Spatial Tasks

In this article, cognitive architecture structures are introduced which address the general perceptual abstraction and general irreducibility problems. These structures support *simulative imagery*. In this section, simulative imagery is explained at a high level, details will be covered in later sections.

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 cases that make general perceptual abstraction difficult.

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. 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 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 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. 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. The relevant aspect of this is that imagery operations simulating the behavior of the agent's low-level controllers is an essential part of the technique. In many cases, the actual controllers used for external action can be run with simulated inputs to allow this (e.g., Leonard et al., 2008).

In using imagery, the problem is 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 Section 4.

Section 2 - Architecture for Spatial Problems

The examples in the previous section informally present some benefits of imagery processing. Here, a theory for an architecture incorporating these aspects is described, specific functional benefits afforded by the theory are presented, and a computational instantiation of the theory is described.

2.1 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.

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 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 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 an interpretation of $\boldsymbol{P}_c$, but can also be manipulated. In particular, it can be manipulated via imagery 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

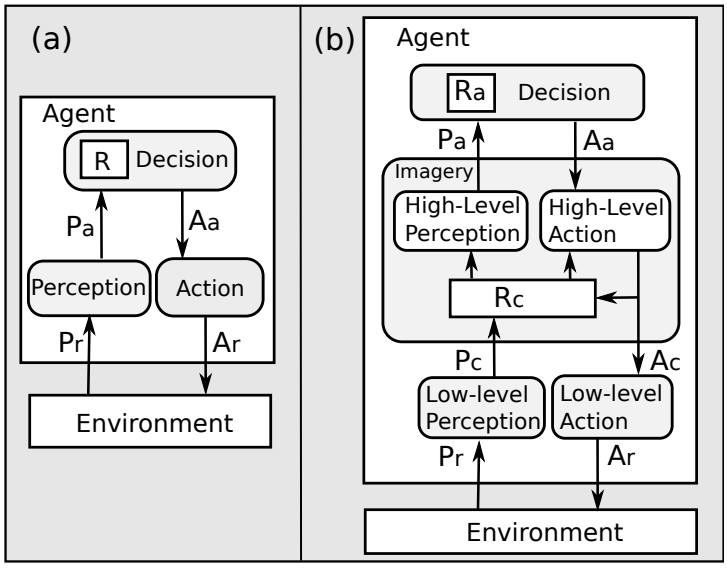

Figure 6: A basic non-imagery architecture (left) and an imagery architecture

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, detailed 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 $\boldsymbol{P}_{\mathrm{c}}$.

2.2 Benefits of the Theory

This architectural theory provides several benefits. Here, we will focus on three of the most important, while a larger set is explored elsewhere (Wintermute, 2010a).

The theory allows 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 in the bin), this determination involves both properties of the moving block and properties of objects that are spatially distant 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. The distinctions cannot be easily detected by composing simple “features” of the current visual scene detected in a bottom-up manner.

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 imagined block collides with any other object.

The theory 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 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 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 imagery 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 what imagery operations were performed. Not only can more relevant properties in particular tasks be generated (as the previous benefit covered), but if the imagery processes simulate actions particular to the task, the architecture is able to encode task-specific properties while retaining a task-independent high-level perception system.

As in the pegged blocks world example, in this work we focus on using simulative imagery 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.

The theory allows symbolic reasoning over continuous processes, eliminating the need for symbolic characterization of controller performance, mitigating the irreducibility problem.

As the action system has access to the concrete representation, low-level controllers can be used which have access to the detailed state information therein. While

irreducibility in some problems can be handled entirely by encapsulating appropriate behaviors in low-level 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, beyond simply supporting low-level controllers, in the theory here, abstract symbolic processing in the agent can reason over controllers without having a characterization of their behavior in terms of abstract states. If the performance of a controller can be simulated with imagery, the agent can 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.

2.3 The Soar/SVS architecture

In this section, the design and capabilities of the Spatial/Visual System, or SVS, is presented. Together with the existing Soar architecture (Laird, 2008), Soar/SVS constitutes an implementation of the theory presented in the previous section. The SVS system has been under development for several years, and a comprehensive overview of the architectural design has been published elsewhere (Lathrop et al., 2011)[3].

[3] Lathrop et al. also discussed the theoretical motivation for the architecture, as is the focus here. While some of the issues presented in this article were encountered there, that discussion was brief and example-based, while this work provides a substantial elaboration and a grounding in general principles.

The overall design of Soar/SVS as it pertains to this work is shown in Figure 7. 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. Agents in Soar can be instantiated to use many different techniques to make decisions, including planning and 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 as shown in the diagram; these different components will be explained shortly.

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.

SVS contains a spatial short-term memory, the spatial scene, which serves as the concrete representation in this work. The spatial scene encodes a set of labeled polyhedrons in two- or three-dimensional space. Additionally, the system has a visual short-term memory, encoding 2D pixel-array information. The visual system is not shown in the diagram, but is discussed elsewhere (Lathrop et al., 2011). It is an alternate

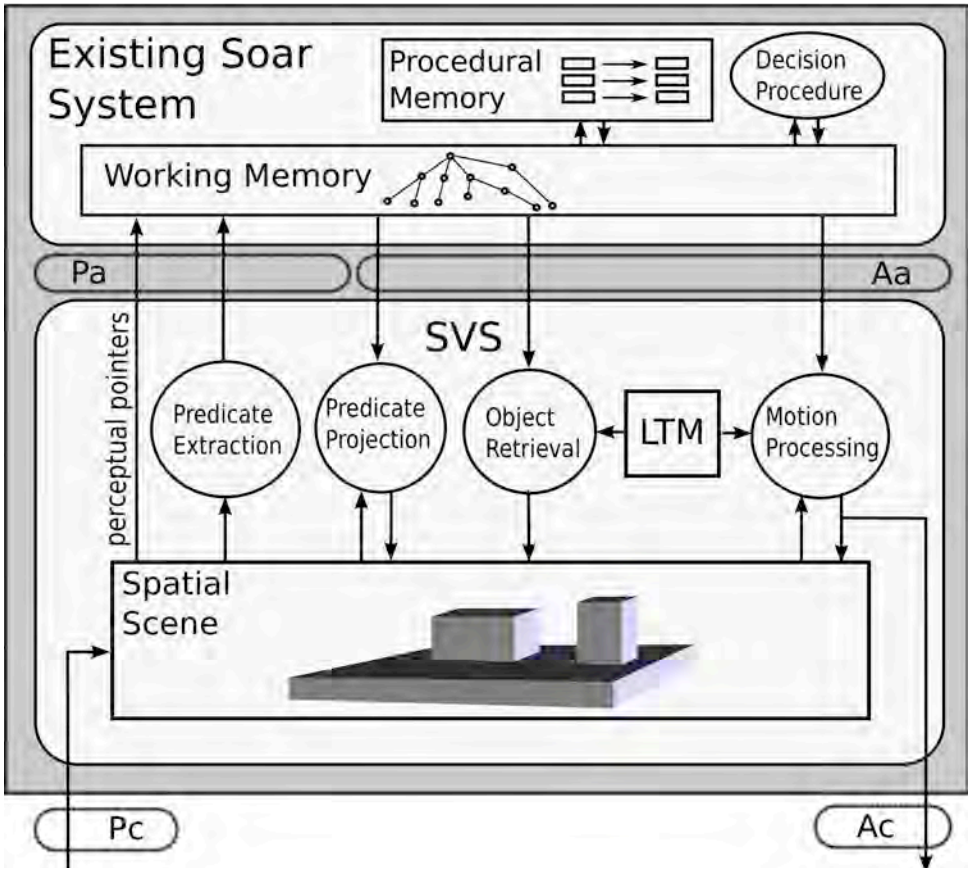

Figure 7: SVS system design. Boxes are memories, circles are processes.

manifestation of a concrete representation with different processing characteristics than the spatial representation, however these differences are not relevant here. A long-term memory is also present in SVS for visual, spatial, and motion data, called perceptual LTM.

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. As we do not address the veridical perception problem, the system does not attempt this. Rather, a task-specific low-level perception system must directly provide the spatial state of the world spatial via $\boldsymbol{P}_c$.

From the point of view of the decision system, the only aspects of the underlying spatial state available are what are encoded in $\boldsymbol{P}_a$. Here, that includes the perceptual pointers, along with information available through predicate extraction processes. A perceptual pointer is a unique token that refers to a specific underlying visual or spatial structure (e.g., an object in the spatial scene). SVS always presents a set of perceptual pointers in $\boldsymbol{P}_a$, and if a pointer appears in $\boldsymbol{A}_a$, SVS can dereference it to access the corresponding perceptual structure.

Predicate extraction processes derive 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. Along perceptual pointers, predicate extraction processes instantiate the high-level perception system in Figure 6. The 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. Images, once created in the spatial scene or visual buffer, are thereafter treated identically to structures in those memories built by perception. To perform imagery, the system needs mechanisms through which spatial images can be created and manipulated. In SVS, there are three such mechanisms. These mechanisms include memory retrieval, which instantiates objects from long-term memory into the spatial scene, and predicate projection, which creates spatial objects based on qualitative descriptions created by Soar (such as "a line between **A** and **B**") (Wintermute & Laird, 2007).

Another important imagery type is a motion simulation (Wintermute & Laird, 2008). A motion simulation process transforms the position of an object in the scene based on some method specific to they type of motion in question. The individual motion types are considered task knowledge, rather than a fixed part of the architecture[4]. Using this

[4] In the current system, motions are implemented by arbitrary C++ functions. Defining a more uniform representation and a learning mechanism is an area for future work.

system, the architecture can support detailed simulative imagery, even when the consequences of an action cannot be described in terms of qualitative predicates or created by retrieving a memory (as the other two mechanisms support).

One aspect of the theory presented in the previous section is that low-level actions share common mechanisms with imagery, and in SVS this commonality is present in the motion simulation system. While some motion processes might not be tied to actions, such as might be used to predict the path of a bouncing ball, other motion processes can include processes where a motion is simulated with the help of an action controller.

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. 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, as will be discussed below.

The imagery components of SVS, along with controllers such as the car controller discussed above, make up the high-level action system in the SVS instantiation of the imagery architecture in Figure 6.

Section 3 - Reinforcement Learning Agents in Soar/SVS

To aid in evaluating the imagery 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 here, so that all control knowledge is learned as opposed to programmed. The aims of this discussion are threefold: to provide further explanation of the theory through demonstrations of implemented agents, to provide evidence of the benefits outlined in Section 2.2, and to connect this work with concepts and related work in the area of reinforcement learning.

3.1 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. 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 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.

The MDP formalism can provide an objective measurement of what it means to have a good state representation for a task: such a 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 1.2. 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 vary 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.

To make a more compact learning problem, allowing faster learning, state aggregation can be used. 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 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 Section 2, 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, and to allow task-specific abstract properties to be encoded by a task-independent perception system.

3.1.1 The Pedestal Blocks World

To provide a concrete example of these benefits in a reinforcement learning setting, the pedestal blocks world task (Figure 2) outlined in Section 1.2 will be used. Here, the agent is presented with a table and three blocks. There are five 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 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 the pixels on the screen, button choices must be output.

3.1.2 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.

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 based on that information, 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 (Wintermute, 2009a). 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.

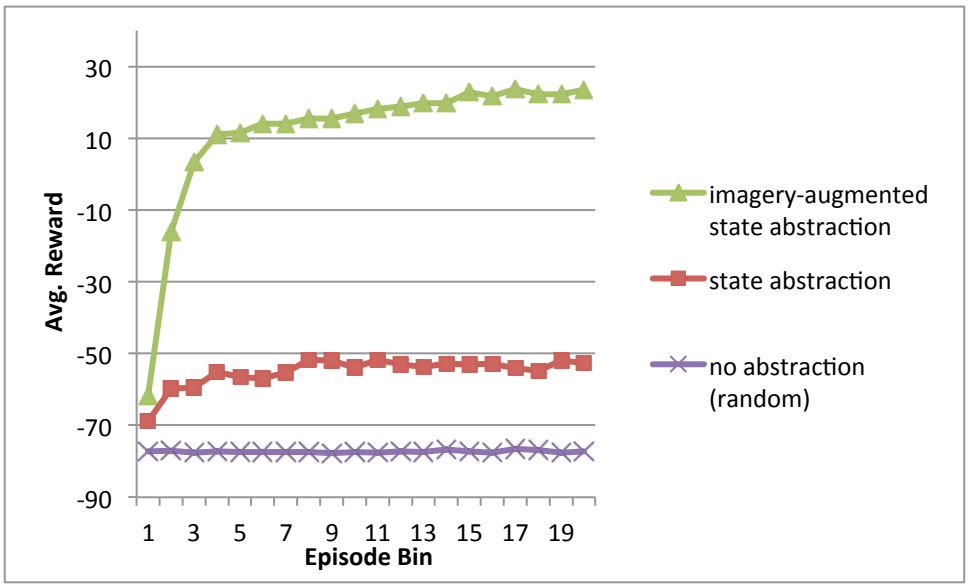

Figure 8: Results of learning in pedestal blocks world showing advantage of imagery-augmented state abstraction.

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 two of the three benefits of the theory.

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).

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 where repeated states would be extremely rare. Figure 8 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 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 provide evidence that the relevant aspects of the architecture are working to mitigate the perceptual abstraction problem.

3.2 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. However, there is room for improvement with this algorithm. The agent uses imagery to make predictions about the consequences of its actions, however, those predictions are treated as properties of the current state, and not as properties of the actions the agent is choosing between. For example, if the agent infers **`collision_if_moved(B,pedestal2)`**, it does not differentiate that this information is relevant to the action of moving B to pedestal2, and not as relevant for the other actions under consideration. The action modeling knowledge implicit in these predictions is not leveraged.

While simulative imagery is performing action modeling, standard techniques for integrating action modeling and RL (model-based RL techniques) are not appropriate here. This is because the result of a prediction, from the perspective of the decision system, is non-deterministic. There is not enough information in the abstract state to reliably predict what the outcome of an action will be, even though that information may be present in the concrete state of the imagery system. This is a problem for model-based RL techniques applied to the abstract state space, which typically rely dynamic programming: the assumption that predictions made for a given state remain applicable when that state is later encountered is false.

Based on these insights, a new technique for integrating reinforcement learning with imagery was developed, ReLAI (Reinforcement Learning with Abstraction and Imagery; Wintermute, 2010b). 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 that a state-action pair belongs to is determined by the predicted next abstract state that will result from it. The aggregate of a state-action pair is called a *category*, and indicated by $C(s,a)$. The abstract state corresponding to concrete state s is indicated by $A(s)$. When ReLAI predicts correctly, then, $C(s,a) = A(s')$.

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 9.

```
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 9: The ReLAI algorithm as instantiated in Soar/SVS.

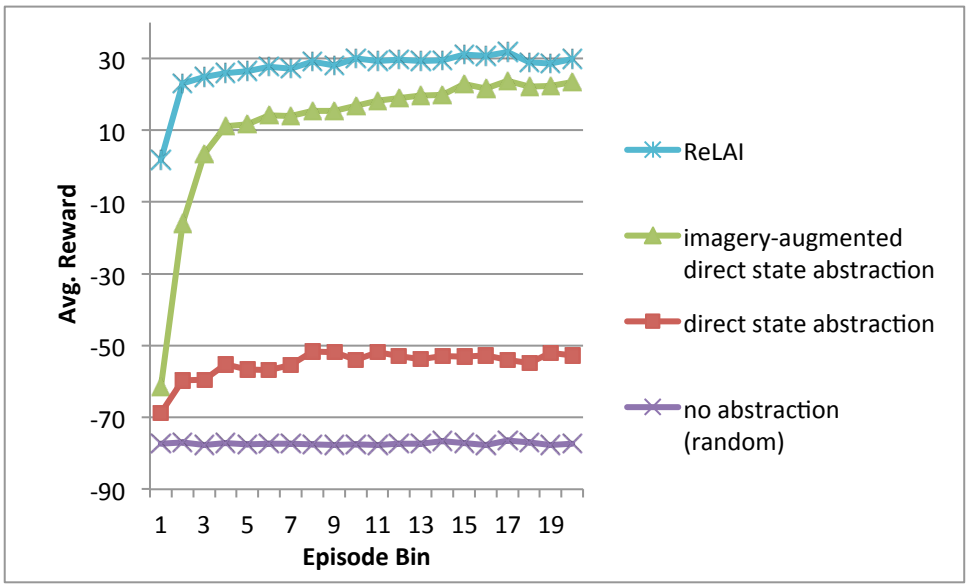

Figure 10: Results of learning in pedestal blocks world showing advantage of ReLAI.

Figure 10 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 action modeling knowledge implicit in its imagery operations. 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.

3.2.1 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 (Wintermute, 2010b). The result of this analysis is a set of three conditions that must hold for guaranteed convergence.

First, predictions must be correct: in all cases $C(s_t, a)$ must be what $A(s_{t+1})$ would equal if action a were to be taken. 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.

Second, the reward received for a transition must always be independent of (s, a), given the next abstract state:

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

Finally, the next abstract state must be independent of the previous (s, a) pair, given the current abstract state and action[5]:

$$Pr\{A(s_{t+1}) = x|s_{t-1}, a_{t-1}, A(s_t), a_t\} = Pr\{A(s_{t+1}) = x|A(s_t), a_t\} \quad (2)$$

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

Even with this reduced constraint, these assumptions can be difficult to match. In the example pedestal blocks world problem as presented above, equation 1 is met, since the reward for a transition is completely determined by the resulting abstract state. However, equation 2 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[6]. Since the agent would learn the optimal policy in the alternate

[5] There is a minor aspect necessary for the proof not captured here: Equation 2 must hold under all possible policies, not just the policy actually followed. There might be some abstraction function that, when used with a particular policy, meets Equation 2 for the actions taken, but would not have for other actions. That possibility will not be considered here.

[6] 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. In the alternate problem, since

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.

3.2.2 ReLAI and Perceptual Abstraction

The theoretical background here can add to the understanding of the imagery benefit of task-specific abstract property generation.

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. 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.

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, where Q^* is the value Q-learning would converge to given enough experience in the unabstracted problem. However, ReLAI allows convergence with abstraction functions that are not Q^*-irrelevant. For example, the abstraction function used in pedestal blocks world is not Q^*-irrelevant. All initial states of the problem are grouped together in a single abstract state, regardless of whether moving **A** to

---

pedestal dimensions change after each move, this correlation is not present. As is apparent in the data, this minor difference does not substantially affect learning.

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[7]. 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 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. Overall, this amounts to strong support 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.

3.3 ReLAI in a Complex Task

While the pedestal blocks world domain provides a simple example of the application of ReLAI, the algorithm (and the architecture supporting it) can handle much more complex tasks. Inspired by other work using arcade games as a source of AI problems

[7] A simple task where the requirements for ReLAI are more straightforwardly met is presented elsewhere (Wintermute, 2010b).

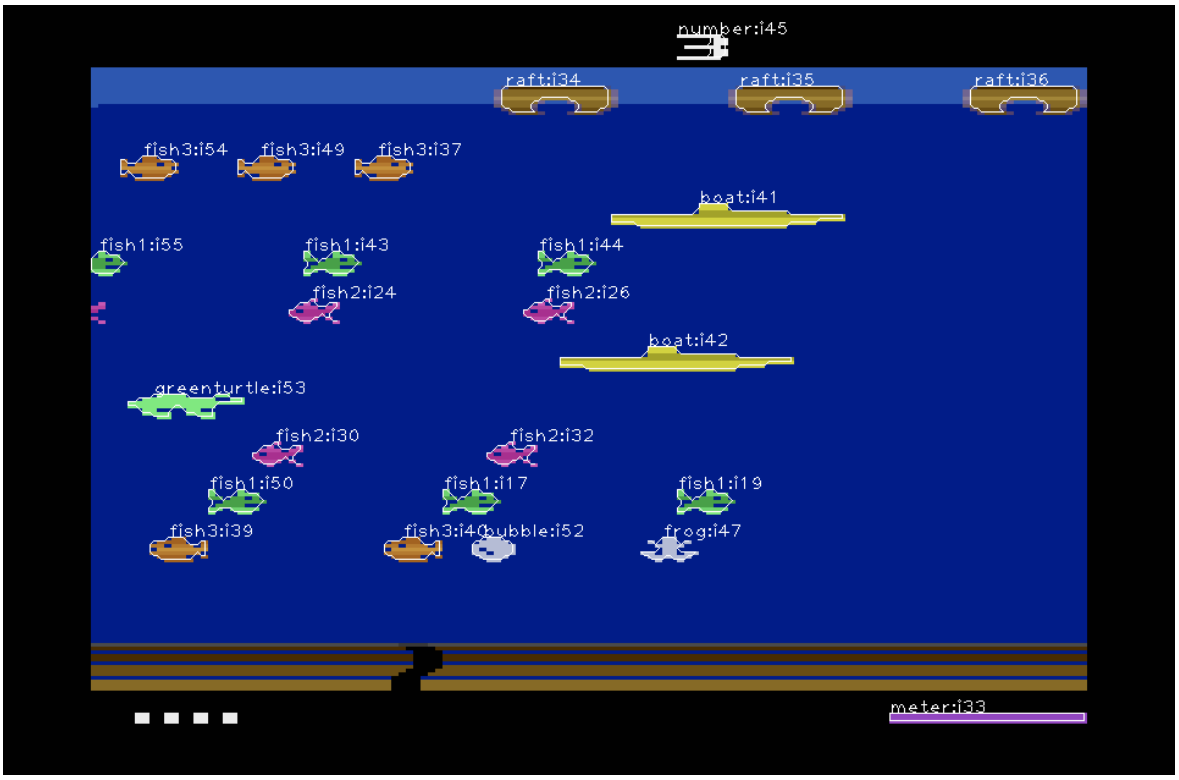

Figure 11: Perceptual information in the game Frogger II, including object labels.

(e.g. Agre & Chapman, 1987; Diuk et al., 2008), an agent has been developed to play the game Frogger II for the Atari 2600 system. The original game is used (run in an emulator) – it has not been reimplemented.

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 2.3.

Figure 11 shows the perceptual information provided by the emulator for Frogger II[8], overlaid with object outlines and category names provided by the 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 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

[8] © Parker Bros., 1983

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.

A ReLAI agent has been created for this task. The agent chooses an action once every 15 game frames (four per second). The experiment here examines the quality of learning that the agent achieves (and not reaction speed), so the emulator is paused while the agent processes the perceptions and chooses an action.

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 a simple 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 in this task encode the following information in working memory:

- a predicate 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 11 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.

To allow for a simple comparison, both a ReLAI agent and a direct state abstraction agent have been created using the same perceptual abstraction. Following the algorithm in Figure 9, 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, 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 direct state abstraction agent's state, so it is not included there. The direct abstraction agent simply infers the abstract state of the spatial scene (which is unmodified by imagery), and uses that as the state signal input to the Q-learning algorithm.

For ReLAI, the requirement that the abstraction captures immediate reward (Equation 1) 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 2) 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 direct state abstraction, 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 collision ns 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.

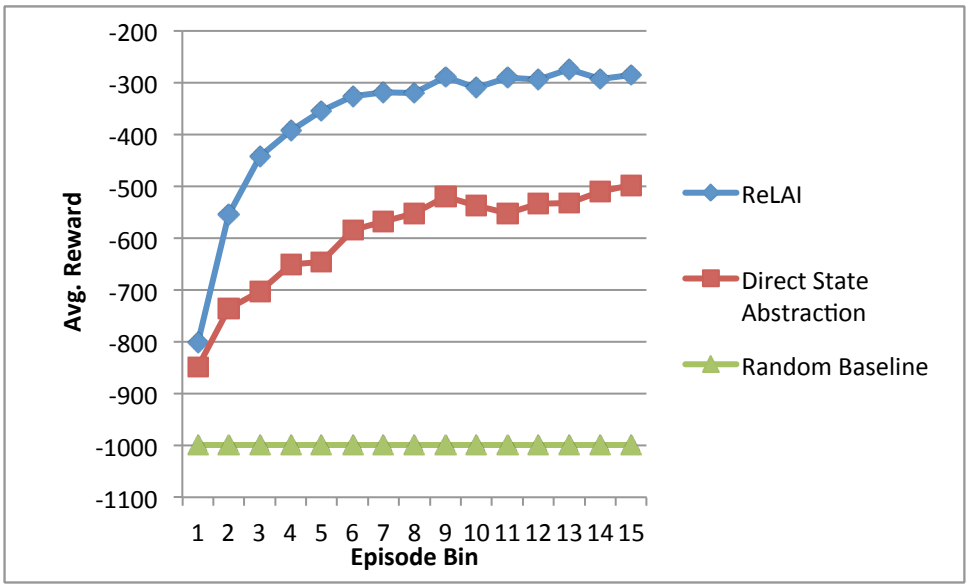

Figure 12: Performance of ReLAI vs. direct state abstraction in Frogger II.

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 12 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. 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 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.

This experiment demonstrates that ReLAI can be empirically useful even when theoretical requirements are not met, and provides evidence that state representations that meet the theoretical requirements of neither direct abstraction nor ReLAI can perform much better with ReLAI. Using the same system, experiments have been

conducted in two further games (Space Invaders and Fast Eddie) with similar results. For brevity, these experiments will not be reported here, but details can be found elsewhere (Wintermute, 2010a).

The ReLAI agent here further demonstrates the first two benefits listed in Section 2.2, which relate to the perceptual abstraction problem. When the 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 about the state that takes into account the precise movement of both the frog and the fish. In that way, simulative imagery is being used to capture movement in terms of abstract symbolic information (the first listed benefit in Section 2.2). Furthermore, the properties involved are based on simulations of the particular task—they are task-specific properties, even though the task-independent SVS predicate extraction system is used to infer them. This is a demonstration of the second benefit in Section 2.2.

Section 4 - Motion Planning in Soar/SVS

As discussed in Section 1.2, 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 4).

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 third benefit in Section 2.2, which allows imagery to mitigate the irreducibility problem,

4.1 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 13: The RRT Algorithm.

The tree is constructed by the algorithm in Figure 13, and Figure 14 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.

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

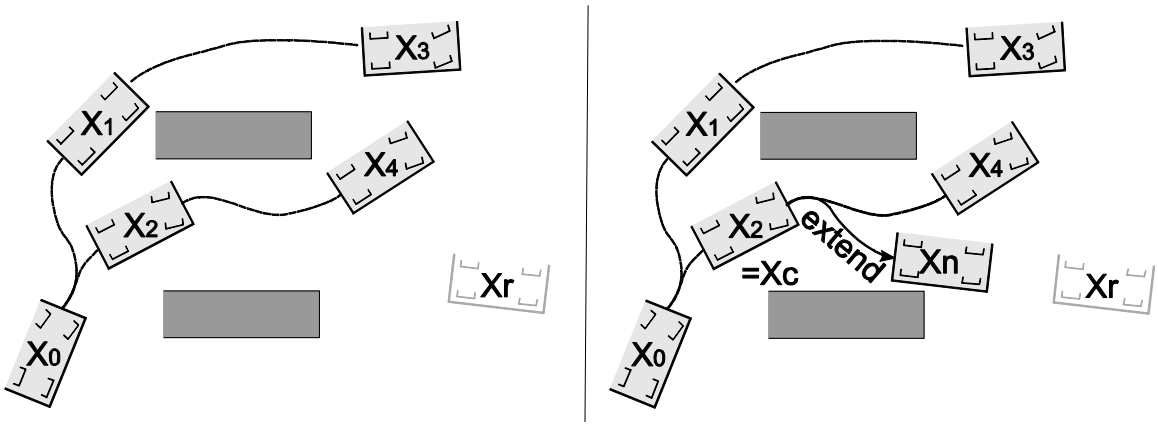

Figure 14: An example of RRT applied to car motion planning.

front of the source. On the left of Figure 14, 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 14. 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 14, 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.

4.2 RRT in Soar/SVS

A version of the RRT algorithm has been instantiated in a Soar/SVS agent. 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 a dynamical systems approach to cognitive science can be tightly integrated with a symbolic AI framework. In addition, it performs well.[9]

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

[9] In other work, the local obstacle-avoiding controller here was directly compared (with favorable results) to a similar controller that simply steers towards the goal (Wintermute, 2009b).

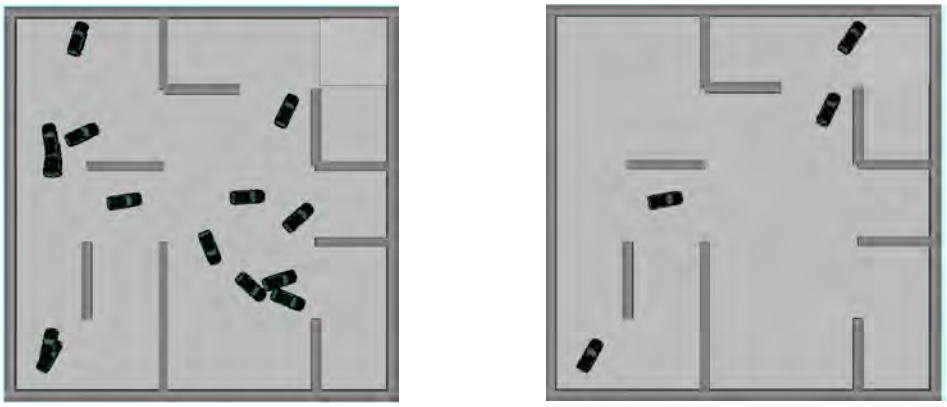

Figure 15: States of SVS Spatial Scene during RRT planning. The problem is to drive a car from lower-left to upper-right. Left: RRT tree, just before a solution is found. Right: Sequence of car positions that solve the problem.

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 13. As a metric for node distance, Euclidean distance was used, with the condition that the distance is infinite where the goal is not in front of the node. 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 15. 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 (creating a new car object in the scene), and this simulation is stepped until a certain time is reached, the goal is reached, or Soar detects a collision

with an obstacle[10]. In all but the last case, the termination of the simulation results in a new node being added to the tree. 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 15 and other similar scenarios. For example, in the problem in the figure, 100 trials were run, and a solution was found after an average of 12 tree expansions. This agent serves as an existence proof that the algorithm can be implemented in Soar/SVS, and as a demonstration of the architecture applied to this task.

4.3 Perceptual Abstraction and Irreducibility in Motion Planning

It is interesting to note that the algorithm here was developed independent of any broad architectural theories, but instead to address a practical need. That indicates that the approach is fundamentally valuable, its utility is not, for example, an artifact of the symbolic assumption in Section 1.1, 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 2.2. Most prominently, it demonstrates the benefits related to irreducibility. 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 low-level 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.

The use of low-level control alone is insufficient to mitigate irreducibility in this situation, though. The task cannot be reduced to abstract symbolic reasoning about controllers (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,

[10] In the implemented system, steering angle and velocity are assumed to reset to zero between nodes, since SVS only retains shape and position information in the scene. Minor enhancements to the architecture allowing motion model instantiations to be preserved would be needed to allow the agent to find paths that are truly continuous in these quantities.

as may be possible when planning the motion of simpler robots. 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 (the third benefit in Section 2.2). 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 first two perceptual abstraction related benefits. The use of simulative imagery allows motion to captured symbolically and allows task-specific abstract properties to be captured (for example, "if the controller is used to seek towards the goal from this state, it will succeed").

Section 5 - Related Work

This paper touches on many areas of research in artificial intelligence, cognitive science, and robotics. In this section a few of the most relevant connections that have not been examined deeply elsewhere in the paper will be discussed.

5.1 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 in principle be adapted to SVS, but this has not been done, as the interface to symbolic processing in Soar has changed. 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 has a different approach to this interface, which is elaborated in a technical report (Wintermute, 2009a). SVI's equivalents to the predicate extraction, predicate projection, and memory retrieval systems in SVS are also simpler, and SVI has no direct equivalent to the motion processing system in SVS, either for imagery or control.

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. The integration of spatial and symbolic states is conceptually different 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 also has no direct equivalent to the motion processing system of SVS, and does not include three-dimensional processing. 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).

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.

5.2 Robotic Systems

Systems developed to support intelligent robotics often address many of the issues discussed here. 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 Section 4. Along with other robotics systems using similar algorithms, this system can be considered as implementing part of the theory, although these aspects are considered more as engineering details rather than theoretical commitments.

Other systems are more directly posed as general-purpose theories. 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 Section 1.4.

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 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 the 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. However, this system does not appear to have been extended beyond its navigation task.

5.3 Visual Routines

The use of a concrete representation as an intermediate in visual processing has been examined in work with visual routines. As stated by Ullman (1984), "The general proposal is that using a fixed set of basic operations, the visual system can assemble routines that are applied to the visual representation to extract abstract shape properties and spatial relations". The argument behind visual routines is very similar to the arguments presented here regarding imagery processing as a means to perceptual abstraction—that manipulation of a concrete representation can allow a larger class of abstract features to be inferred.

An important difference between Ullman's work and that presented here is that visual routines are not regarded as imagery processes, but rather as an intermediate stage of perception. An important implication of this difference is apparent in the pedestal blocks world results shown in Figure 10. The graph shows a performance difference between an agent that knows that certain abstract perceptual features are the

implications of particular proposed actions (the ReLAI agent), versus an agent that has applied the same visual routines, but does not associate the results with particular actions (the imagery-augmented direct state abstraction agent). Essentially, manipulating the concrete representation via imagery allows the agent some built-in knowledge about the results of that manipulation—if the agent has chosen to imagine action A, it knows the resulting abstract information is an effect of action A, rather than a more generic property of the current state, information which it can use to its advantage. Furthermore, imagery operations result in a persistent concrete state which can serve as the basis of further manipulations. This ability is used in the Frogger agent. At each time step, the results of actions are simulated first by imagining the motion of objects not controlled by the agent (e.g., the motion of the fish), and then sequentially overlaying that state with the imagined consequences of each particular action (the motion of the frog in each direction). Without a persistent imagery representation, this decomposition would not be possible.

5.4 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 Section 3 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. In the resulting system, an agent experiencing a completely new state can leverage knowledge learned in other states that are nearby in terms of the components of the state.

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).

There is a concern here, since a core 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 of simulative imagery, but also incorporates function approximation. Presumably both of these aspects are important for the performance of TD-Gammon, and it can be concluded that in this case simulative imagery and function approximation are at least partially complementary.

5.5 Imagery in Psychology

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 efficiency characteristics, supporting the hypothesis of depictive imagery.

The examination of the perceptual abstraction and irreducibility problems can further inform 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. 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.[11] As stated by Forbus (1993), in reference to work in qualitative spatial reasoning,

> "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."

The irreducibility problem, along with its proposed solution in the form of low-level controllers 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

[11] 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.

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 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.

Section 6 - Conclusion

The overall goal of this work has been to make progress 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, 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, and the Soar/SVS architecture, which follows the theory, was introduced. Several agents running in this architecture were examined, demonstrating the benefits of the underlying theory. These benefits, and the evidence supporting them, will be summarized here:

The theory allows 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 Frogger agent, 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.

This benefit derives from the ability for the concrete representation in the architecture to be locally manipulated by imagery. 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 manipulated, it is difficult to see how the agent could infer this long-term reachability information.

The theory 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 all of the agents, both individually and collectively. The SVS architecture includes a fixed, task-independent high-level perception system, which each of the agents use to capture task-specific abstract properties. In each case, using task knowledge, 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 high-level perception system, achieving progress towards general perceptual abstraction.

The theory allows symbolic reasoning over continuous processes, eliminating the need for symbolic characterization of controller performance, mitigating the irreducibility problem.

This benefit was demonstrated by the RRT agent. As is discussed in Section 4.3, the motion-planning task here is fundamentally irreducible. Low-level controllers simplify the action space of the agent, and simulative imagery of 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. As a result, the agent is able to act intelligently in an irreducible task.

Overall, the goal of this work 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.

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* (Vol. 278).

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

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.

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

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*. Presented at the International Conference on Cognitive Modeling, Pittsburgh, PA.

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 the 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.

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*. Presented at 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.

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.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence, 2*(3-4), 189–208.

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.

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.

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

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*.

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. Presented at the 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* (pp. 224-235). Amsterdam: IOS Press.

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. doi:10.1016/0921-8890(91)90017-F

Langley, P., Laird, J. E., & Rogers, S. (2009). Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, *10*(2), 141-160. doi:10.1016/j.cogsys.2006.07.004

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. (2009). Extending Cognitive Architectures with Mental Imagery. In *Proceedings of the Second Conference on Artificial General Intelligence*.

Lathrop, S. D., Wintermute, S., & Laird, J. E. (2010). Exploring the Functional Advantages of Spatial and Visual Cognition From an Architectural Perspective. *Topics in Cognitive Science*, no-no. doi:10.1111/j.1756-8765.2010.01130.x

Lathrop, S. D., Wintermute, S., & Laird, J. E. (2011). Exploring the functional advantages of spatial and visual cognition from an architectural perspective. *Topics in Cognitive Science*, *to appear*.

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.

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). doi:10.1002/rob.20262

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.

Nason, S., & Laird, J. E. (2005). Soar-RL: integrating reinforcement learning with Soar. *Cognitive Systems Research, 6*(1), 51-59. doi:10.1016/j.cogsys.2004.09.006

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

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).

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. doi:10.1.1.18.8192

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. doi:10.1.1.51.4764

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. doi:10.1016/S0364-0213(99)80026-3

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. doi:10.1145/203330.203343

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*. Presented at the The 2010 International Conference on Intelligent Agent Technology.

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

Wintermute, S. (2009b). Integrating Action and Reasoning through Simulation. In *Proceedings of the Second Conference on Artificial General Intelligence* (pp. 192-197). Presented at the AGI-09, Amsterdam - Beijing - Paris: Atlantis Press.

Wintermute, S. (2010a). *Abstraction, Imagery, and Control in Cognitive Architecture* (PhD Thesis). Ann Arbor: University of Michigan.

Wintermute, S. (2010b). Using Imagery to Simplify Perceptual Abstraction in Reinforcement Learning Agents. In *Proceedings of the the Twenty-Fourth AAAI Conference on Artificial Intelligence* (pp. 1567-1573). Menlo Park: AAAI Press.

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 (AAAI-07)* (pp. 1572-1577). Presented at the AAAI-07, Vancouver, BC: AAAI Press.

Wintermute, S., & Laird, J. E. (2008). Bimodal Spatial Reasoning with Continuous Motion. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)* (pp. 1331-1337). Presented at the AAAI-08, Chicago, IL: AAAI Press.