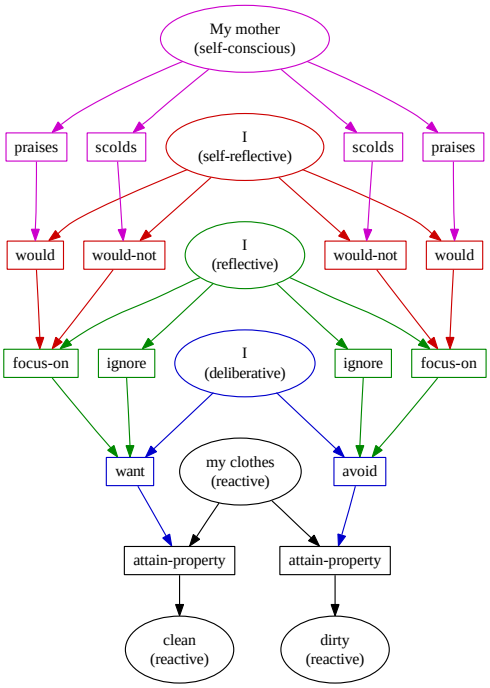
BO MORGAN

A SUBSTRATE FOR ACCOUNTABLE LAYERED SYSTEMS

A SUBSTRATE FOR ACCOUNTABLE LAYERED SYSTEMS

BO MORGAN



Ph.D. in the Media Arts and Sciences

Media Laboratory
Massachusetts Institute of Technology

August 2011

Though there be no such thing as Chance in the world;
our ignorance of the real cause of any event
has the same influence on the understanding,
and begets a like species of belief or opinion.

— David Hume

Dedicated to the loving memory of Pushpinder Singh.

1972 – 2006

ABSTRACT

A system built on a layered reflective cognitive architecture presents many novel and difficult software engineering problems. Some of these problems can be ameliorated by erecting the system on a substrate that implicitly supports tracing of results and behavior of the system to the data and through the procedures that produced those results and that behavior. Good traces make the system accountable; it enables the analysis of success and failure, and thus enhances the ability to learn from mistakes.

I have constructed just such a substrate. It provides for general parallelism and concurrency, while supporting the automatic collection of audit trails for all processes, including the processes that analyze audit trails. My system natively supports a Lisp-like language. In such a language, as in machine language, a program is data that can be easily manipulated by a program. This makes it easier for a user or an automatic procedure to read, edit, and write programs as they are debugged.

Here, I build and demonstrate an example of reflective problem solving in a block building toy problem domain. I then apply my approach to a simulation of life in a rigidbody physical environment in order to show scalability to non-trivial problem domains. In my demonstration multiple agents can learn from experience of success or failure or by being explicitly taught by other agents, including the user. In my demonstration I show how procedurally traced memory can be used to assign credit to those deliberative processes that are responsible for the failure, facilitating learning how to better plan for these types of problems in the future.

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

Morgan, B.; “Moral Compass: Commonsense Social Reasoning Cognitive Architecture”; http://em-two.net/about; Commonsense Tech Note; MIT Media Lab; 2011 January

Smith, D. and Morgan, B.; "IsisWorld: An open source commonsense simulator for AI researchers"; AAAI 2010 Workshop on Metacognition; 2010 April

Morgan, B.; “A Computational Theory of the Communication of Problem Solving Knowledge between Parents and Children”; PhD Proposal; MIT Media Lab 2010 January

Morgan, B.; “Funk2: A Distributed Processing Language for Reflective Tracing of a Large Critic-Selector Cognitive Architecture”; Proceedings of the Metacognition Workshop at the Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems; San Francisco, California, USA; 2009 September

Morgan, B.; “Learning Commonsense Human-language Descriptions from Temporal and Spatial Sensor-network Data”; Masters Thesis; Massachusetts Institute of Technology; 2006 August

Morgan, B.; “Learning perception lattices to compare generative explanations of human-language stories”; Published Online; Commonsense Tech Note; MIT Media Lab; 2006 July

Morgan, B. and Singh, P.; “Elaborating Sensor Data using Temporal and Spatial Commonsense Reasoning”; International Workshop on Wearable and Implantable Body Sensor Networks (BSN-2006); 2005 November

Morgan, B.; “Experts think together to solve hard problems”; Published Online; Commonsense Tech Note; MIT Media Lab 2005 August

Morgan, B.; “LifeNet Belief Propagation”; Published Online; Commonsense Tech Note; MIT Media Lab; 2004 January

Don't do anything that isn't play.

— Joseph Campbell

ACKNOWLEDGMENTS

I will put my acknowledgments here.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

GPS General Problem Solver

RMDP Relational Markov Decision Process

FSM Finite State Machine

Part I

INTRODUCTION

1

LEARNING TO ACCOMPLISH GOALS

> Problem-solvers must find relevant data. How does the human mind retrieve what it needs from among so many millions of knowledge items? Different AI systems have attempted to use a variety of different methods for this. Some assign keywords, attributes, or descriptors to each item and then locate data by feature-matching or by using more sophisticated associative data-base methods. Others use graph-matching or analogical case-based adaptation. Yet others try to find relevant information by threading their ways through systematic, usually hierarchical classifications of knowledge—sometimes called "ontologies". But, to me, all such ideas seem deficient because it is not enough to classify items of information simply in terms of the features or structures of those items themselves. This is because we rarely use a representation in an intentional vacuum, but we always have goals—and two objects may seem similar for one purpose but different for another purpose.
>
> — Marvin Minsky

In this introductory chapter I will give an overview of the problem of reflectively learning to accomplish goals. First, I will develop the assumption that any process that I choose under given constraints can be reflectively monitored and useful information, e.g. beginning and ending times, will be temporarily stored. In other words, I assume that my system natively supports procedural reflection for all processes. Given this assumption, I show in this chapter that the problem of building a system that learns to accomplish goals in its environment becomes a simple loop-less causal structure involving at least four different types of learning. These four types of learning complete a circle of causal learning from goals through actions to physical states and back to goals, but because of time, causality is still in an ordered lattice structure beginning with goals as the causal roots for all other knowledge.

In later chapters I will introduce experiments I have performed in different physical problem simulations. The first is a simple proof-of-concept demonstration of my reflective learning algorithm in a blocks world problem domain. Also, I show my reflective architecture extended to a larger physical reasoning domain, including multiple agents learning to cook together in a kitchen environment. In this larger problem space, agents can communicate high-level procedures specified in a simple programming

language that is shared by the agents, and refers to both mental and physical actions of the agents.

1.1 THE AGENT-ENVIRONMENT MODEL

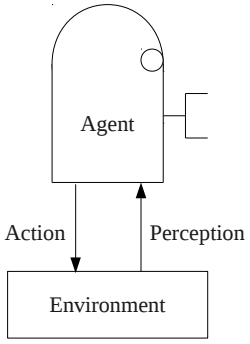


Figure 1: The agent-environment model.

Figure 1 shows the basic agent-environment model. In this model, I make a distinction between the environment and the agent. At any given time, the agent and the environment are each represented as a specific static form of data. Further, these representations change over time, according to a given transfer function. I will treat this system as a deterministic system, although one could imagine adding random variables to the transfer function: the basic theory is the same. It is easier to add randomness to a deterministic theory than the opposite. There are also many benefits to developing a deterministic model with perhaps a pseudorandom aspect because this allows for the repeatability of scientific experiments, for which the model may be used as a metric. The two processes communicate information along two channels: (1) an action channel from the agent to the environment, and (2) a perception channel from the environment to the agent.

1.1.1 *The State-Action Model*

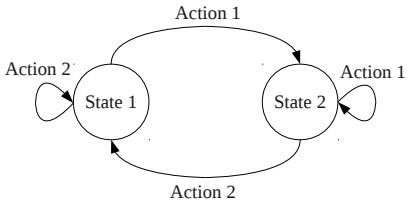


Figure 2: The state action model. Two states are represented by nodes and two actions are represented by edges from each of the two states.

The agent is in an environment, which is in a specific state. My agent performs an action, which can affect the state of the

environment. Figure 2 shows a simple Finite State Machine (FSM) state-action model, which has two states for the environment and two actions for the agent to perform in each of these states. The state-action model is a simple model for how actions map to changes in the state of the environment.

1.1.2 *A Multiple Agent-Environment Model*

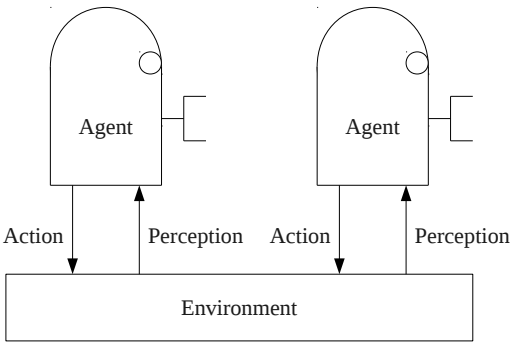


Figure 3: The multiple agent environment model.

In order to model social intelligence, we introduce the multiple agent environment model shown in Figure 3.

1.1.3 *Agent Process Communication*

Because agent processes can only directly act on and perceive the environment, all communication between agent processes must occur through the environment process. We assume that agents can communicate some form of symbolic knowledge structure in the absence of noise. These representations are used to communicate processes from one agent to another. Specific examples of such a process representation that we have used in our experiments are described in more detail in Sections 4.3.2–4.3.3.

1.1.4 *A Relational State Representation*

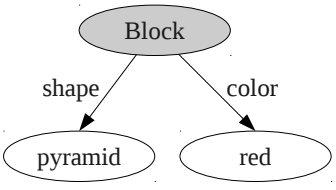


Figure 4: Frame-based relational representation.

See Figure 4.

1.1.5 *Introducing Reflection Early in the Process*

Now, I have introduced my problem as being divided into at least two processes, at least one agent process and an environment process. Further, I have introduced how a basic process may be thought of as an FSM. At this point, I introduce a model for keeping track of the changes in a computational process: reflective knowledge. I purposefully make this assumption before I define the details of the agent model process because, in my approach, I would like to reflectively reason about potentially any aspect of this agent process.

1.1.6 *Reflective Knowledge*

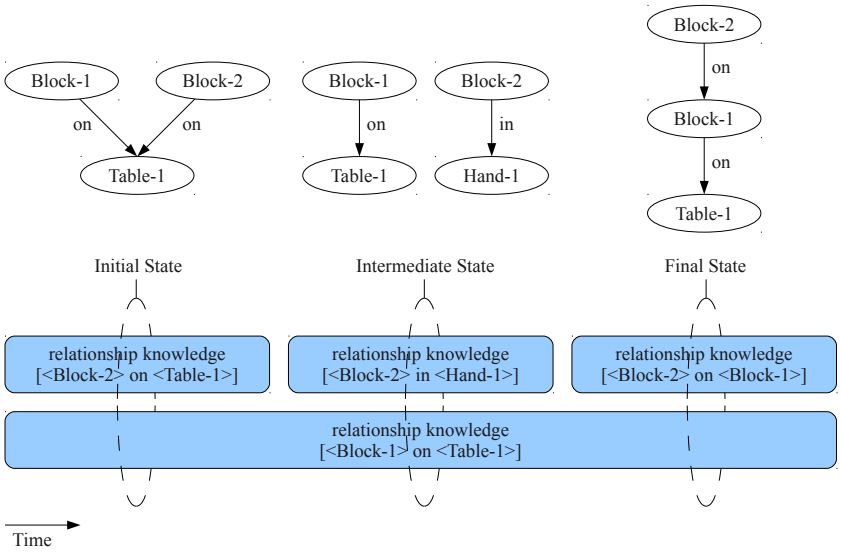


Figure 5: A reflective event representation shows the changes in a labeled graph.

While the term meta-knowledge is used to describe the very general idea of knowledge about knowledge, I use the term reflective knowledge to refer to the specific type of meta-knowledge that refers to knowledge about the changes to a knowledge structure. If I keep track of the changes to a knowledge structure, I can later integrate these changes in order to obtain an equivalent form of that knowledge structure as it was at any point in the past.[1]

1.1.7 *Frame Perceptions*

See Figure 6.

1 See Section H.3 for a discussion of more realistic models of computation, including multiple-core and cluster models.

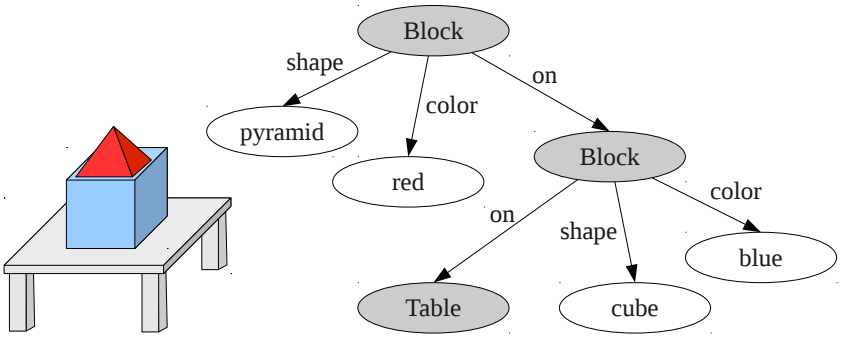


Figure 6: Collections of frames used as perceptual input to agent.

1.1.8 *Partially Observable State Model*

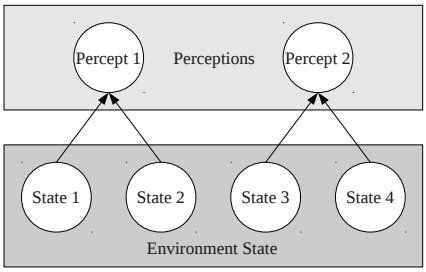


Figure 7: The partially observable model.

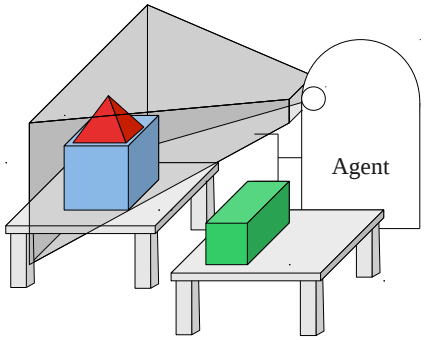


Figure 8: The state of the environment is only partially observable.

The agent process does not have complete access to the state of its environment. The agent's perceptual stream of information depends on the state of the environment, but it is a function of the environment and not the actual state of the environment. In other words, the perceptual state that is communicated from the environment to the agent is an injective function mapping the environment to the perception of the agent. See Figures 7 and 8 for two examples of partially observable environments.

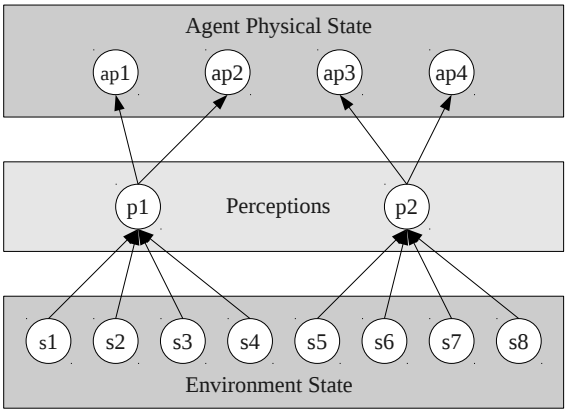


Figure 9: The agent has an abstract physical model of the environment.

1.1.9 *Agent Abstract Physical Model*

See Figure 9.

1.1.10 *A Physical Goal Representation*

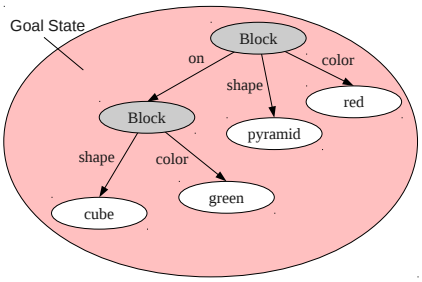


Figure 10: A physical goal representation is a structural relationship that may or may not currently exist within the physical knowledge base.

See Figure 10.

1.2 THE REFLECTIVE LEARNING CYCLE

I have now introduced the definitions for my relational state space, along with how reflective tracing, given certain assumptions, allows the creation of reflective event representations of the action resources within the agent's mind.

In this section, I will describe how these representations and reflective event representations can be used to reflectively learn the different types of knowledge used for planning.

Figure 11 shows the abstract four-step learning to plan cycle. Planning is an arbitrarily complicated process, but with these four steps I have created basic semantic divisions between the types of knowledge involved in reflectively learning to plan. These basic divisions in the learning process can be summarized as:

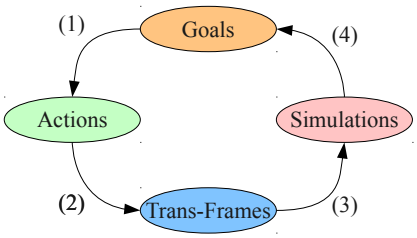


Figure 11: Learning to plan four step cycle.

1. Changing goal knowledge initiates causal learning.
2. Physical precondition concept for goal occurrence is refined.
3. Physical transition concepts for simulating action resource are refined.
4. Physical concept for goal occurrence is refined.

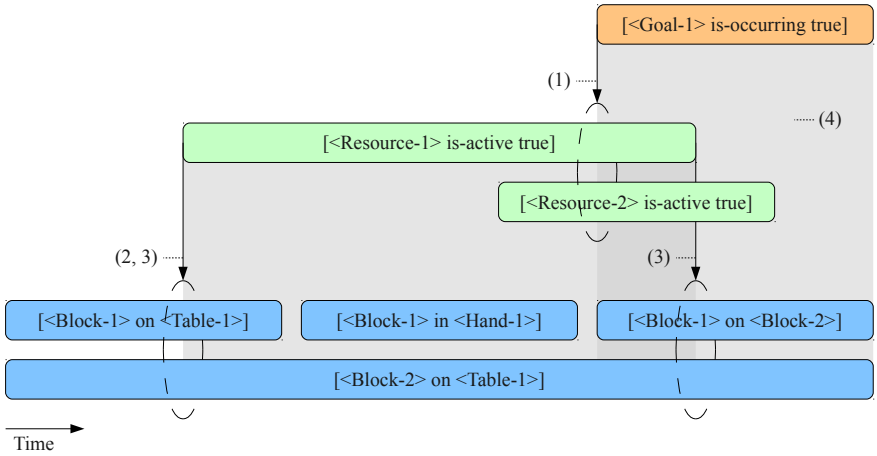


Figure 12: The reflective learning cycle is a way to learn knowledge used to plan toward goals that maintains the causal dependency structure of the knowledge.

A system without a goal has no reason to learn the effects of its actions. Figure 12 shows how this process of learning causal models relating reflective states to physical states is a fundamentally goal-driven process. We show how this goal prediction process leads to at least a four step causal chain of knowledge learning. It is important to point out that although the learning process is a cycle over time, this ends up creating a causal structure for knowledge dependency without loops—a critical property in general to maintain for any causal representation.

These four goal-oriented cyclical learning steps build four of the major types of knowledge that my goal-oriented inference and planning system uses. Let us now briefly introduce these four types of knowledge before going into a more thorough explanation of the utility of each.

1. Goal-oriented action hypotheses: focuses and constrains the initial action learning search.
2. Action physical precondition goal occurrence hypotheses: allows predicting whether or not an action will accomplish the given goal under a given physical precondition.
3. Action physical precondition trans-frame hypotheses: allows for physically simulating the given action in a given physical state.
4. Physical hypotheses for predicting goal occurrence: allows for predicting if a given physical state implies that the goal is occurring.

Note that there are two different concepts of time that are displayed in Figure 12. It is important to not confuse these. The first kind of time is represented by the sequence of change events in the knowledge structure that was reflectively traced in order to generate the temporal event representation shown in the figure. This first time is represented and labelled as the horizontal axis in the figure. The second kind of time represented in the figure is represented by the enumeration of the four learning steps. This learning process is a reflective learning process because it learns from reflective knowledge gathered from tracing processes.

In the following four sections, we will describe the utility of each of these four different types of learned knowledge in more detail.

1.2.1 *Causal Action Hypotheses*

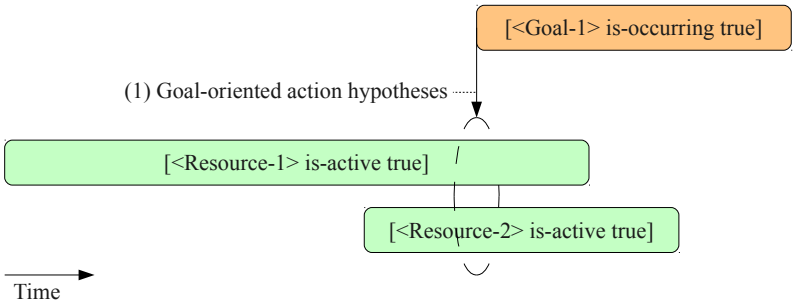


Figure 13: Goal-oriented action hypothesis knowledge.

learning goal: focusing learning on a subset of the state space.

Figure 13 shows the first step of the reflective learning cycle. This first step of the learning cycle is caused reflectively by the goal event beginning to exist. When a goal event comes into existence, a reflective process monitoring this goal knowledge makes a list of potential actions that might be useful for causally predicting this goal condition in the future. In the figure we can see that a simple interval-point intersection operation between the beginning point of the goal event and any action event interval is enough to generate two actions as initial causal hypotheses.

See Section 4.3.5 for details on the techniques that I used for generating goal-oriented action hypotheses in my system.

1.2.2 *Action Goal Occurrence Hypotheses*

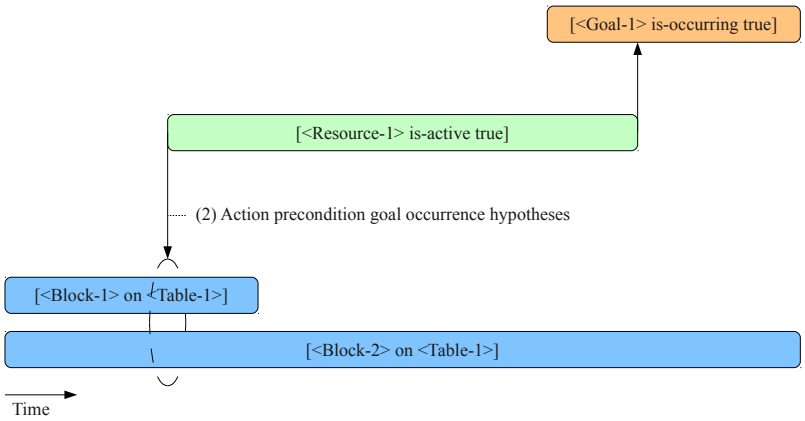


Figure 14: Action precondition goal occurrence hypotheses can be learned from these reflective representations.

Action precondition goal occurrence hypotheses are a form of knowledge that is learned in order to predict whether or not a goal will be occurring if a given action is taken in a given conceptual category of a given abstraction of the state space. Figure 14 shows the reflective event representations that can be used to learn this type of knowledge.

1.2.3 *Action Trans-Frame Hypotheses*

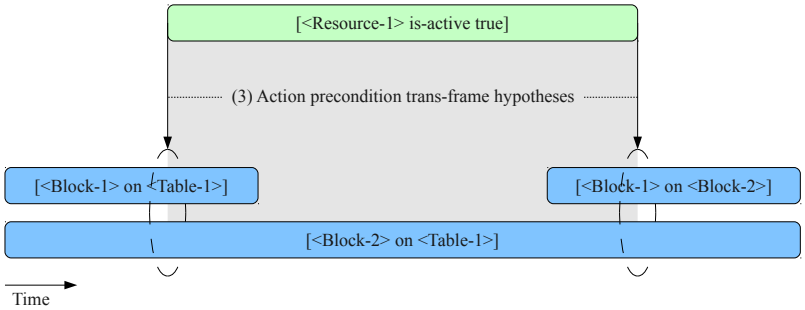


Figure 15: Action precondition trans-frame hypotheses.

See Figure 15.
See Figure 16.

1.2.4 *Physical Goal Occurrence Hypotheses*

See Figure 17.

Remove Events	Add Events
[<Goal-1> is-occurring false] [<Block-1> on <Table-1>]	[<Goal-1> is-occurring true] [<Block-1> on <Block-2>]

Figure 16: A trans-frame for an event is a list of differences in knowledge between the beginning and ending of the event.

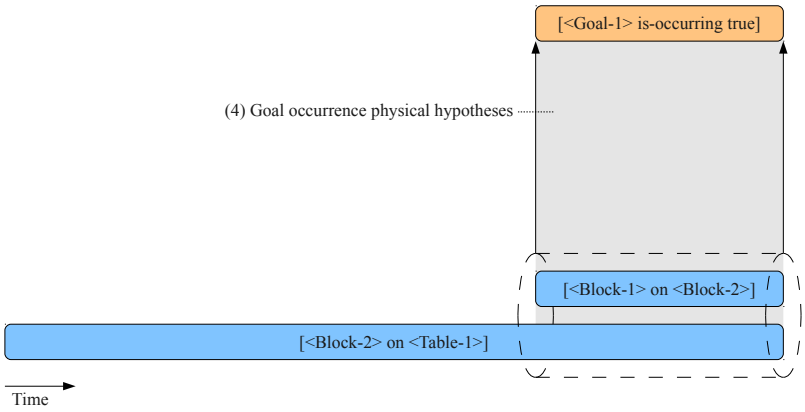


Figure 17: Goal occurrence physical hypotheses.

1.3 THE PLANNING PROCESS

I have now introduced my definition of reflective event representations,

1.3.1 *Partially-Ordered Plan Representation*

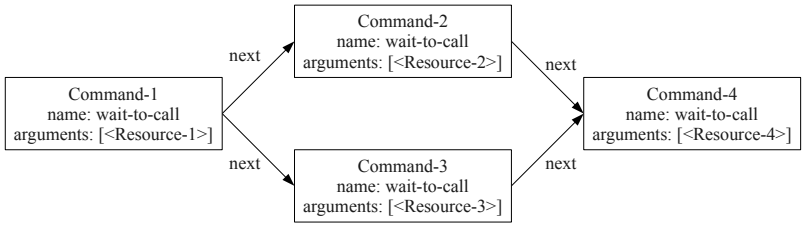


Figure 18: A partially-ordered plan representation containing serial and parallel components.

The partially-ordered plan representation allows a partially-ordered temporal organization for a set of commands. A plan with a branch-and-join control structure is shown in Figure 18.

1.3.2 *Inferring the Effects of a Plan*

The planning process requires an inference algorithm to infer future and past states based on cause-effect relationships between reflective knowledge and physical knowledge. Figure 19 shows a

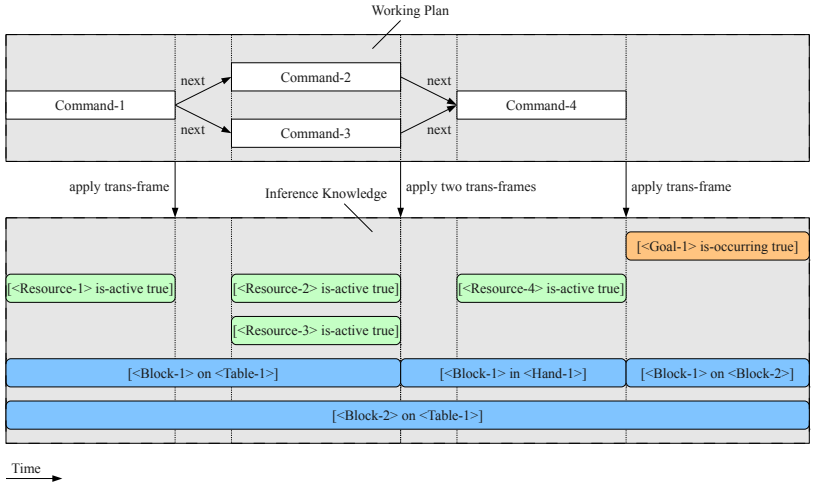


Figure 19: Inferring the effects of a plan.

possible inference for a given plan. There are many ways to plan and the specific planning system that I have implemented for my experiments is discussed further in Section 4.3.4.

1.3.3 *Planning Machine Operations*

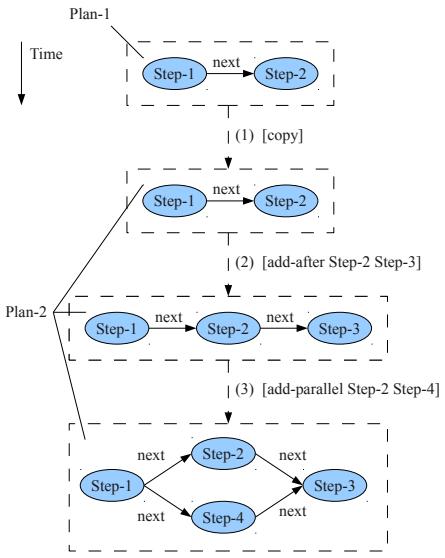


Figure 20: A few planning machine operations.

A few operations for manupulating partially-ordered plans are shown in Figure 20.

1.4 LAYERING REFLECTIVE LEARNING

I have now introduced how reflective representations can be used for learning how to plan. In this section, I will show how using the same reflective approach to learning to plan toward goals can be used to build another layer of reflective control on top of our

planner. This next layer of reflective control learns to accomplish reflective goals for the plan domain of the first-level planner.

1.4.1 *Planning Machine Reflective Knowledge*

The same reflective learning to plan cycle can be applied to the actions of the planning process itself, introducing an idea of reflective goals. The basic four steps of the reflective learning to plan cycle are very similar when applied to the deliberate planning process:

1. Changing reflective goal knowledge initiates causal learning.
2. Deliberative precondition concept for reflective goal achievement is refined.
3. Deliberative transition concepts for simulating deliberative action resource are refined.
4. Deliberative concept for reflective goal achievement is refined.

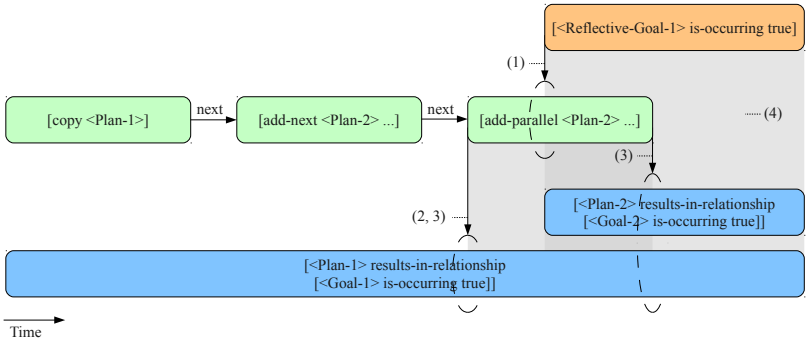


Figure 21: Planning machine reflective knowledge.

See Figure 21.

2 PROBLEMS TO SOLVE

2.1 BUILD A REFLECTIVE KNOWLEDGE SUBSTRATE

The assumption that I introduced in Section 1.1.5, "Introducing Reflection Early in the Process", requires that changes in my knowledge representation can be traced and compiled into other representations, such as the reflective event representations used in learning to accomplish goals.

2.1.1 *Automatic Collection of Audit Trails for All Processes*

Audit trails must be collected so that after the fact the events that any process is resposible for can be used for many types of reflective control purposes, such as learning to plan. Although a system that keeps track of everything that it does would support reflection, the audit trail recorder in a system that does not grow indefinitely in memory consumption would have options for focusing the collection of audit trails, while also allowing for their garbage collection when they are no longer needed by another reflective process.

2.1.2 *General Parallelism and Concurrency*

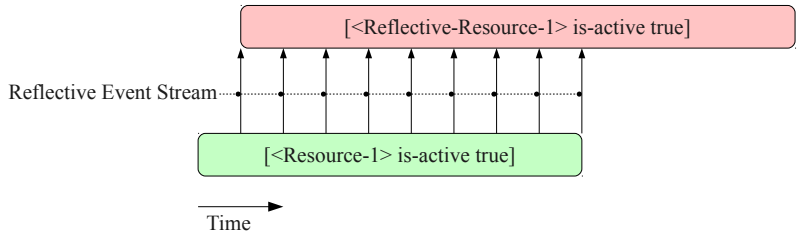


Figure 22: Concurrent parallel reflection efficiency.

Because the procedurally reflective programming paradigm focuses on event streams of changes to memory, there is an inherent ability to express procedurally reflective algorithms in a streaming parallel language.

Figure 22 shows how a simple process can be reflectively monitored without slowing down the fundamental process by more than the constant time factor necessary for generating the event stream. Many different parallel reflective processes can be reflectively processed concurrently without any additional slowdown in the primary problem solving process.

2.1.3 *Program as Data*

The ability for a process to manipulate a program as data makes it easier for that process or a user to read, edit, and write programs as they are debugged. The ability to change the functionality of a program at run-time is a key component to creating an adaptive problem solver. Because of this, a programming language with a run-time compiler is very useful for building these types of adaptive systems that learn to debug and re-program their own subprograms. For purely the reason of developing problem solving learning algorithms, it is critical for a reflective substrate to be able to treat its own programs as data. A compiler is a simple example of a program that must treat a program as data. The planning process is a more general type of compiling problem that also requires learning the effects of primitive actions. So, a system that is planning or learning how to plan, must have some sort of representation of processes that it is interpretting as plans. Compiling simply makes this interpretation step in the planner more efficient, but this is a very convenient efficiency when building practically useful learning systems.

2.2 LAYERED REFLECTIVE PROBLEM SOLVING

2.2.1 *Analogy between Physical Goals and Planning Goals*

2.3 LEARNING BY CREDIT ASSIGNMENT

2.3.1 *Use Reflective Representations for Better Models of Learning*

2.3.2 *Tracing Knowledge Provenance for Credit Assignment of Success or Failure*

3

THEORY AND ALTERNATIVES

3.1 TWO POPULAR APPROACHES TO MODELLING INTELLIGENCE

Recently, there have been two directions of research with the goal of building a machine that explains intelligent human behavior. The first approach is to build a baby-machine that learns from scratch to accomplish goals through interactions with its environment. The second approach is to give the machine an abundance of knowledge that represents correct behavior.

Each of these solutions has benefits and drawbacks. The baby-machine approach is good for dealing with novel problems, but these problems are necessarily simple because complex problems require a lot of background knowledge. The data abundance approach deals well with complicated problems requiring a lot of background knowledge, but fails to adapt to changing environments, for which the algorithm has not already been trained.

3.1.1 *Adaptability in Complex Environments*

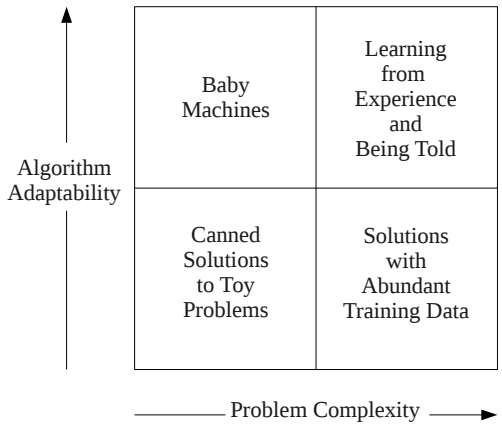


Figure 23: Problem complexity versus algorithm adaptability.

We would like to build intelligent machines that are able to perform household tasks, such as cooking, cleaning, and doing the laundry, but these tasks seem insurmountably complex, containing organically unpredictable events. We would like our machines to expertly handle these extremely complicated problems, and we would also like them to adapt to learn in unexpected or novel situations. One popular approach to building a machine that performs complicated tasks is to give the machine a large training dataset that details every possible situation that the machine may find itself within, along with the correct action in that situation. This is the so-called "supervised" learning approach. These al-

gorithms do not adapt to novel situations well, and collecting these datasets is often impossible for many problems, such as cooking and cleaning because it is too difficult to enumerate all possible situations, in which the machine may find itself. Also, if the machine is cooking a meal, we would like to be able to explain an idea for a new recipe to the machine, or to perhaps be a partner in discovering new recipes, or we may simply want to explain to the machine that a guest has a specific allergy to walnuts, making that ingredient an exception for this meal but not others. Figure 23 shows how problem complexity and algorithm adaptability can be thought of as a two-dimensional space into which different algorithmic approaches can be used as solutions.

3.1.2 *The Abundant Data Approach*

There have been many approaches to modelling complex forms of reasoning by collecting large amounts of knowledge that describes correct or acceptable behavior in a domain. For example, there are examples of complex multi-agent commonsense simulation environments collects thousands of examples of users interacting in a complicated object-oriented social simulation (Orkin & Roy, 2009), (Orkin *et al.* , 2010). These systems have complicated domains, but these projects do not attempt to build agents that attempt to accomplish goals. Instead, these systems are inference systems that simply try to reproduce typical behavior, rather than goal-directed behavior.

There are many commonsense reasoning systems that do not interact with simulation environments at all, but which attempt to demonstrate commonsense reasoning by being told large amounts of knowledge. The Cyc project is one large such project that has been told large amounts of logical knowledge (Lenat *et al.* , 1990). There is also effort directed toward populating Cyc with knowledge automatically gathered from the web (Matuszek *et al.* , 2005). The OpenMind project (Singh *et al.* , 2002) is a project that gathers large amounts of approximately correct commonsense knowledge from people online. The OpenMind knowledge has been turned into many inference systems that can compare and generate new commonsense knowledge (Liu & Singh, 2004b,a; Speer *et al.* , 2008).

3.1.3 *The Common Sense Reasoning Problem Domain*

Common sense is the set of common reasoning abilities shared by most people in a given social group. Another way to say this is that common sense is the set of reasoning abilities that one would assume of a typical person that they meet for the first time and know nothing about. For example, most people have a naive theory of physics, so you would expect someone to know that things fall when they are not supported and liquids flow or are

absorbed unless they are in a container. Common sense relies on a lot of knowledge that is assumed that most everyone knows.

Building a machine that demonstrates common sense reasoning is a long-standing goal of the field of artificial intelligence. One of the difficulties in developing algorithms for dealing with a common sense reasoning domain is that the algorithm needs a lot of background knowledge about a given domain before it can answer even simple questions about it. However, this knowledge is often only true in very specific situations and has many exceptional cases. For example, the knowledge that most birds can fly is generally true, but we also know that many birds are flightless, such as penguins, ostriches, and road runners. Also, we have knowledge about the typical behavior of objects; for example, we know that refrigerators keep things cold, but we also reason efficiently about exceptional cases, such as when the refrigerator is not plugged in, or when the power goes out.

3.1.4 *Representations for Common Sense Reasoning*

There have been many approaches to artificial intelligence that use first-order logic as a representation for these types of knowledge and their exceptions, but these systems become cumbersome in their inability to express "fuzzy" sorts of relationships, such as when the knowledge is applicable, for example the modifiers, "most of the time", "usually", and "almost never", are difficult to express in first-order logic. When we have a lot of knowledge, we need ways to keep track of in which situations this knowledge is useful. This is a form of "meta-knowledge", or knowledge about knowledge. Meta-knowledge about first-order logic cannot be expressed in first-order logic, so another type of representation is required for this type of knowledge. Therefore, we need other ways to represent our knowledge in addition to logic.

> "Nonetheless, theorem proving is in the worst case an intractable problem, even with no variables or unification, so no algorithm is going to work on the problem all the time. In this respect, theorem proving, for all its superficial formalism, is a lot like other branches of AI. Where a theorist views a problem as solved if he has found an efficient algorithm or a discouraging lower bound, an AI researcher is often happy to find an algorithm that seems to work on some interesting problems, even though he doesn't really know the bounds of what it can do. Exploration will reveal the extent of its powers-each time it solves one more interesting problem something has been gained." — Drew McDermott

3.2 COMPARABLE COGNITIVE ARCHITECTURES

EM-ONE, Cyc, Icarus, ACT-r, Soar, and Prodigy are comparable cognitive architectures to the one that I have built.

3.2.1 *The EM-ONE Cognitive Architecture*

We worked with Pushpinder Singh from 1999 to 2006 on the first version of the Emotion Machine architecture, EM-ONE (Singh, 2005). During that period, we discussed that one weakness in the EM-ONE system is its reliance on tracing only the declarative prolog statements, among other necessary but untraced procedural code. Although EM-ONE contained a large amount of procedural knowledge, none of the effects of this procedural knowledge could be debugged reflectively. Toward solving this problem, we have based our approach on a memory layer that can trace the provenance of select memory events.

3.2.2 *Computational Models of Cognition about Cognition*

Cox (2005) gives a good overview of the cognitive sciences that deal with the problem of thinking about thinking, or "metacognition".

3.2.3 *Shades of Belief as Debugging Meta-Knowledge*

Stein & Barnden (1995) applies a powerful prototype system to perform reflective case-based reasoning over "shades" of beliefs in knowledge.

3.3 BOUNDED RATIONALITY

There is an approach of economics and game theory that is called bounded rationality (Simon, 1972). These models deal with the time constraints of not only acting efficiently in a domain but also in optimizing the planning actions involved in accomplishing goals. This approach assumes that there is an absolute numerical reward value for accomplishing goals. In the sense that absolute values for all of the goals of a system are seldom known in practice, the bounded rationality approach is limited in a general sense similar to the simpler reinforcement learning approach.

3.3.1 *Feedback Control Model for Accomplishing a Single Goal*

Now that we have discussed the basic model of learning from experience what good goal states may be from rewards, let us consider the representations for the state space of the perceptions and actions of our model. Control theory has given us many useful models for agents that control continuous environments. For

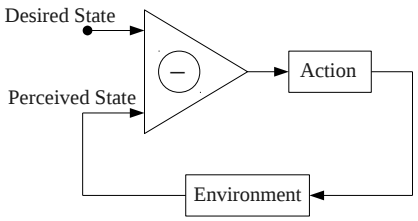


Figure 24: The feedback control model for accomplishing a single goal.

example, Figure 24 shows a simple difference feedback control circuit that is used in simple linear control systems. The system is given a desired state, there is a difference device that calculates the difference between the actual perceived value from the environment, and the control system then executes an action based on that difference, which affects the environment. The result in such a negative feedback loop is that the agent's perception of the environment is closer to the desired state.

3.3.2 *Means-End Analysis*

In 1959, Newell, Shaw, and Simon published a report on a means-end analysis model that was designed to solve any symbolically represented problem (Newell *et al.* , 1959). Their system was called the General Problem Solver (GPS), and worked by being able to work with relational representations of current and desired states. The agent had a catalogue of differences between states that it knew how to minimize. The system worked by finding the largest difference and executing the associated method for reducing this difference. This work has grown into the Soar model (Newell, 1990) for better solving symbolic planning problems, and dealing with impasses for when the planning search runs out of options.

3.3.3 *Difference-Engine Model for Accomplishing Multiple Goals*

(Minsky, 1988, p. 78)

3.4 MACHINE LEARNING

One encompassing goal of the field of machine learning is to develop systems that can accomplish goals.

For example, a Markov Decision Process contains a transfer function, which is basically a combinational device.

Of course, this combinational device would get more complicated if we added probability.

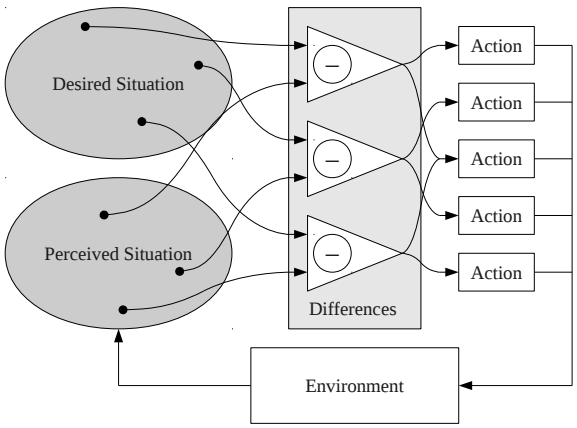


Figure 25: The difference engine model for accomplishing multiple goals.

Part II

MY APPROACH

4

A SYSTEM

4.1 SYSTEM OVERVIEW

Building a machine that demonstrates the general intelligence required for commonsense reasoning is a longstanding goal of the field of artificial intelligence. There have been many approaches to building a machine that demonstrates general intelligence, some are based on logical representations, others are based on large collections of statistical knowledge, while still others approach the problem by learning everything from scratch from the physical world. I see the problem as requiring a combination of many different types of representations and reasoning processes.

4.1.1 *Reflectively Traced Frame Memory*

Procedural tracing can be thought of as enabling a part of an interpreter that checks for important events as a process runs. This ability can be used to trace the temporal order of events, i.e. generating a trace, or to otherwise organize or summarize events in a more useful way that can be used by other procedures, either concurrently or after the fact. Having this ability built into the memory system, allows keeping track of the provenance of information as it is written and read. This ability is built into all structures in the architecture, so if any tracing of data provenance is required at any time for any object, this ability can be enabled.

At higher levels, these procedural tracing events result in event streams that can be listened to by multiple concurrent processes that each allocate iterators for a stream. As a concurrent process increments its iterator, it reasons about the event that has occurred in the object memory, and the result of this reasoning is the creation of meta knowledge in a causally consistent knowledge base. This is an example of the "glom" theory of cognitive evolution, where cognitive abilities are added on top of previous cognitive abilities without changing the underlying functionality. I've used procedural tracing to maintain multiple consistent representations for the same knowledge.

When a plan fails, I need to correct the knowledge that generated that plan. When multiple processes are adding knowledge to the same knowledge base, it becomes important to keep track of the provenance of this knowledge, when I need to make distinctions between situations that appear identical. If I need to learn a new rule for categorization of situations, or even a new category entirely, I need to go back to the correct features and processes that performed those categorizations of the identical situations that I need to further distinguish.

4.1.2 *Efficiency Problems with Reflective Tracing*

There are serious efficiency problems that must be carefully sidestepped when one is dealing with reflective knowledge. For example, infinite reflection loops result in an infinite processor and memory consumption pattern, after only a single change to a knowledge representation. The solution to this problem is to have various means of controlling the reflective tracing focus. I will discuss the various methods for controlling the tracing focus of the reflective memory that I have found useful in the development of my system in Section 4.1.3.

4.1.3 *Methods for Focusing Reflective Tracing*

Section 4.1.3 is referenced from Section 4.1.2.

4.1.4 *Procedural Tracing*

The idea of having procedural tracing at the operating system level is important because it does allow all programs running on the operating system to assign credit to processes when bugs do occur.

Although it is important to have protected memory boundaries between programs for reasons of security, privacy, and stability, having good ways for processes that do share memory to trace the provenance of individual memory events, allows for much tighter and intelligent interaction between all processes in the entire system.

4.1.5 *Trace Only an Appropriate Level of Abstraction*

It does not make sense to trace below a certain depth of processing. If I were to trace all of the lowest level details of execution at all times, there would be no way to take advantage of that amount of detailed information.

There are barriers in my system for only allowing tracing to occur for specific parts of the code. The ability to focus the tracing of object usage allows the possibility of tracing either high or low-level events in a uniform manner.

4.1.6 *Keeping Tracing from Taking Too Much Time and Storage*

When a set of objects are out of the focus of tracing, these objects run operate at full speed and do not increase the memory usage of the tracing component. There is a new proposal by McCarthy to build a new programming language that remembers everything that it does; it is called Elephant 2000 (McCarthy, 1994).

4.2 AN OPERATING SYSTEM AND A PROGRAMMING LANGUAGE

I've written a multi-core operating system, including a compiler and a programming language, on top of this traceable and distributed memory layer (Morgan, 2009).

Because the system is meant to combine many artificial intelligence techniques, I have tested my platform running thousands of parallel processes concurrently on an 8-core machine. These processes can control and watch one another execute. I feel that the field of AI is not separate from the low-level details of software engineering, and my project embodies that philosophy.

Many people see AI as being a purely theoretical and mathematical field, and I strongly disagree. I need good software engineers in order to solve most of the problems I face in getting these massive software systems to work together.

I have built a layered cognitive architecture on top of my custom operating system, programming language, and compiler.

4.2.1 *Why not use Lisp?*

Lisp is a great programming language. I wrote a custom programming language for the project and didn't use lisp. Lisp simply isn't fast enough, and isn't very well supported; when you find a bug in a lisp compiler, it is difficult to find the support to fix the bug. I wrote the first version of the reflectively traced memory system in lisp and realized that Steele Bourne Common Lisp had memory bugs when the system grew beyond 600 megabytes of RAM. Allegro Lisp is a commercial solution, but it costs many hundreds of dollars for their commercial compiler, and I feel strongly against having that commercial requirement for building academically intentioned open-source software. The main problem with lisp is it's lack of speed and lack of support, so I found myself writing a lot of C extensions even when programming in Lisp. C is good for speeding up inner loops of algorithms as well as necessary for interfacing with the Linux, Mac, or Windows operating systems, which are all written in C.

4.3 A LAYERED COGNITIVE ARCHITECTURE

Further, I have developed a cognitive architecture within my language that provides structures for layering reflective processes, resulting in a hierarchy of control algorithms that respond to failures or other events in the layers below.

4.3.1 *Perceptual Support of Physical Knowledge*

See Figure 26.

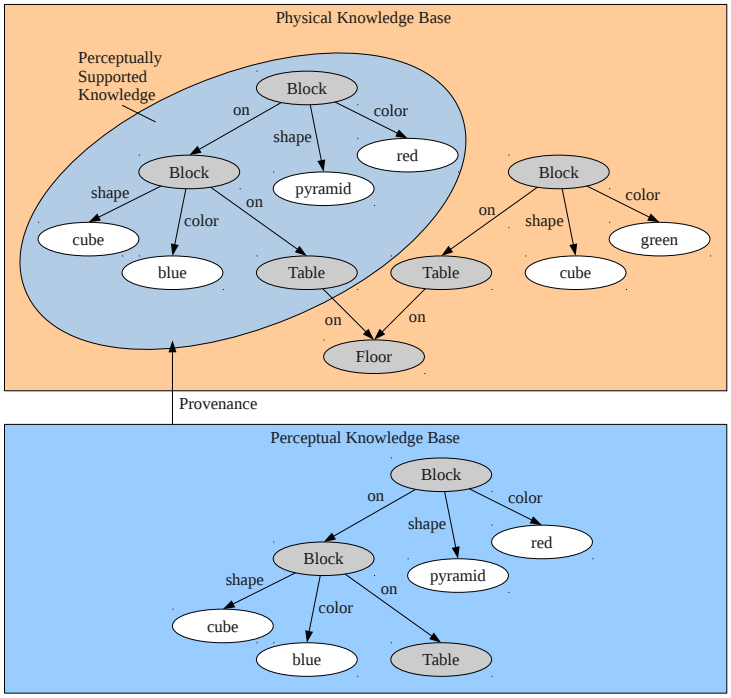


Figure 26: Perceptual provenance provides support for physical knowledge.

[prog	[pick-up	'red	'cube]
	[put-on	'blue	'cube]
	[pick-up	'green	'pyramid]
	[put-on	'red	'cube]]

Table 1: Representation of a serial process.

4.3.2 *A Serial Process Representation*

A representation of a serial process is shown in Table 1. This representation is an ordered tree symbolic representation, which is capable of representing any Lisp-like expression in my language interpretter.

4.3.3 *A Parallel Process Representation*

See Table 2.

[prog	[parog	[use-left-hand-to-pick-up	'red	'cube]
		[use-right-hand-to-pick-up	'green	'pyramid]
	[parog	[use-left-hand-to-put-on	'blue	'cube]]
		[use-right-hand-to-put-on	'red	'cube]]]

Table 2: Representation of two serial parallel processes.

4.3.4 *Details of Inferring the Effects of a Plan*

4.3.5 *Goal-Oriented Action Hypothesis Generation Techniques*

4.3.6 *Debugging Plans by Reflectively Tracing the Provenance of Knowledge*

Section 4.3.4 is referenced from Section 1.3.2.

Section 4.3.5 is referenced from Section 1.2.1.

I trace the procenance of knowledge from perceptual knowledge as it is manipulated into other knowledge representations, driven by a goal-oriented learning algorithm. Normally, when plans fail, rule learning is used to update beginning and ending condition transition function hypotheses for actions relative to various knowledge bases.

In many systems, when these plans fail, it is often unclear which part of the plan was responsible for the failure. In the simplest cases, a specific low-level action may have been executing, which implies that precondition categories were incorrectly mapped to postconditions or the range of postconditions was not broad enough. In the worst cases, it is impossible to tell what part of the plan failed because the plan is such a complicated tangle of compiled numbers that the only recourse is to nudge a few of these numbers using hill-climbing toward an average of a universal goal. Rethinking this problem with my new approach that maintains provenance for knowledge used in creating plans allows debugging the complex web of goal-oriented knowledge, and reflectively, also manipulating the processes involved in creating that knowledge.

5 EXPERIMENTS

5.1 BLOCKS WORLD AS A SIMPLE REAL-TIME SYMBOLIC CONTROL PROBLEM DOMAIN

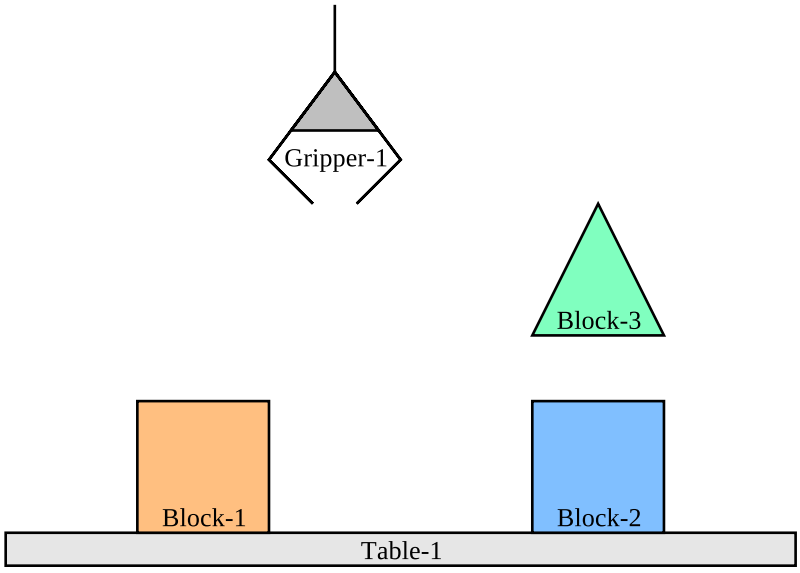


Figure 27: Blocks world is a simple real-time symbolic control problem that I use to demonstrate my reflective control learning theory.

I use blocks world, a canonical toy AI problem, in order to demonstrate my example of reflectively learning to plan. See Figure 27 for a screenshot of my blocks world problem physical simulation.

See Table 3 for an example set of perceptual input that corresponds with the physical situation shown in Figure 27.

5.1.1 *Working in a World of Building Blocks*

In his PhD thesis, Terry Winograd worked in the world of building blocks (Winograd, 1970). This program maintained traces of its goals and subgoals, which enabled it to answer questions about why it performed certain actions. This system worked because it stored goals.

Knowing the goal state of the computation is important, and I do not ignore this aspect in tracing the deliberative layer. My system is able to answer these sorts of questions, as this simply requires climbing the stack of mental resource activations, but when debugging the deliberative process, it is helpful to not only

[Gripper-1 is me]	[Gripper-1 movement_command []]
[Gripper-1 is-a gripper]	[Gripper-1 color black]
[Gripper-1 is-holding []]	[Block-1 is-a block]
[Block-1 color brown]	[Block-1 shape cube]
[Block-1 on Table-1]	[Block-1 left-of Gripper-1]
[Block-2 is-a block]	[Block-2 color blue]
[Block-2 shape cube]	[Block-2 on Table-1]
[Block-2 right-of Gripper-1]	[Block-3 is-a block]
[Block-3 color green]	[Block-3 shape pyramid]
[Block-3 right-of Gripper-1]	[Table-1 is-a block]
[Table-1 color white]	[Table-1 shape cube]
[Table-1 left-of Gripper-1]	

Table 3: Blocks world agent perceptual input.

know the ending point of computation but also the means toward that end.

5.1.2 *Terry Winograd's SHRDLU and Goal Tracing*

I am building upon what was learned from Winograd's thesis (Winograd, 1970) in terms of using traces of the deliberative process as well as using a semantic model of the world in order to understand communications between agents. I have chosen to use a simpler and more direct language interface between agents that refers more directly to the semantic information and mental processes involved. I have experimented with implementing the original SHRDLU english language parser, although I believe the parsing process can be better controlled as a goal-oriented set of concurrent processes than as the stack-based depth first search that I started writing in my initial experiments.

5.2 A PHYSICAL SIMULATION OF A KITCHEN AS A SOCIAL COMMONSENSE REASONING DOMAIN

I have experimented with applying my cognitive architecture to a larger social commonsense reasoning domain with parents that teach children as they attempt to accomplish cooking tasks in a kitchen. See Morgan (2011) for details about my six-layered reflective theory of social and moral reasoning, which assumes the existence of a procedurally reflective infrastructure with a reflective problem solver written within it.

5.2.1 *Why Not Work Solely Within the Blocks World Domain?*

The building blocks approach is a good precedent. However, there are many problems with only demonstrating a solution

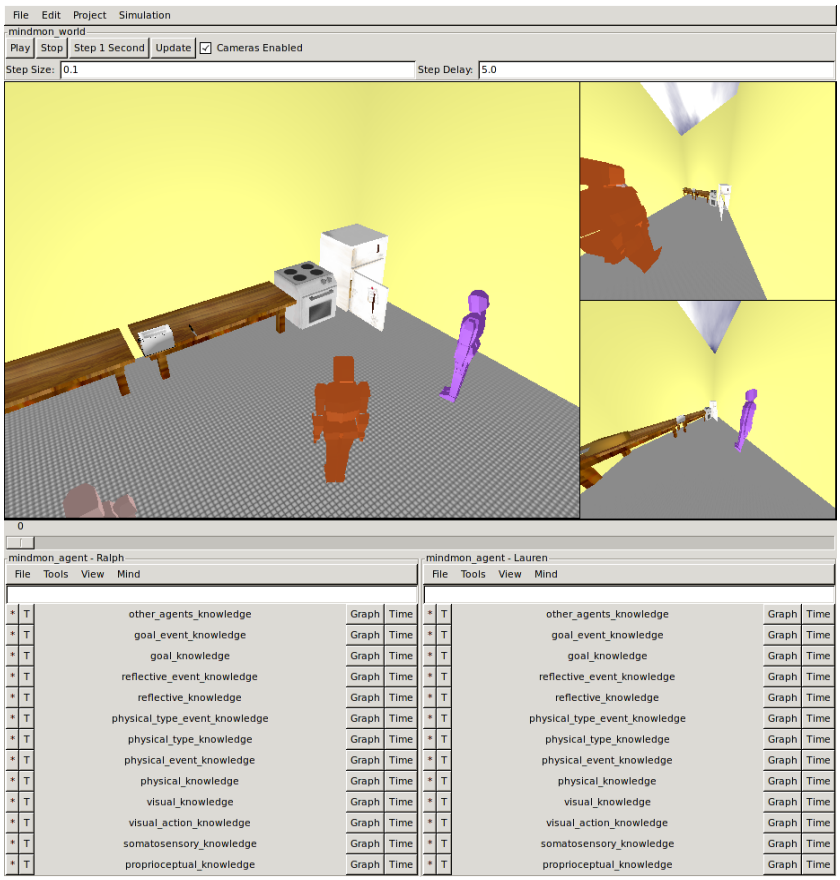


Figure 28: Isis World is a larger real-time symbolic physical simulation, which I use to demonstrate that my reflective goal-oriented learning approach scales to the physical and reflective problem spaces of slightly more complicated learning problems.

on a toy problem. First, an approach demonstrated to solve a small problem, often do not scale to larger problem domains of similar complexity. So, I feel that it is important to show the same reflective approach to learning can also be applied to a domain with a much larger state space than the toy blocks world problem. I then, have shown the theoretical gains of my approach by using the canonical model as a tool for explanation, and now I show that my model does scale to larger problem domains of only slightly more complexity. See Smith & Morgan (2010) for a discussion of the benefits of approaching the social commonsense reasoning problem with a physical simulation of a kitchen.

I have conscripted my domain of object types in the kitchen, such that it is currently comparable to the number of object types that Winograd used in his thesis. My object types do have different ways that they may be used, which is a small addition of complexity. Although I do not introduce many of the complexities of ontological reasoning, a common approach to commonsense reasoning, e.g. Cyc (Lenat *et al.* , 1990), my system demonstrates an important new approach to commonsense reasoning that grounds learning by being told in the domain of goal-oriented reasoning, which allows organizing and debugging knowledge in terms of what goals it is useful for accomplishing.

5.2.2 *Why is Cooking in a Kitchen a Good Problem to Model?*

Smith & Morgan (2010) discuss reasons for why focusing on solving cooking tasks in a kitchen will lead to a better understanding of the reflective control of multiple agents in a common complicated social environment. Further, Morgan (2010) describes how focusing on this domain from a reflective architectural point of view can lead to future models of self-reflective and self-conscious learning between parents and children.

From an educational perspective, Dewey (1907) has theorized that kitchens are a good example of a rich learning environment for children. Some form of kitchen, or a social place where food is cooked for a family, is ubiquitous across cultures. Kitchens have a clear production goal, namely food. Many mental realms must be used in accomplishing even simple cooking goals, including: math, physics, chemistry, thermodynamics, language, society, family, imprimer learning, concurrent planning, etc. See Figure 28 for a screenshot of my graphical user interface to the cognitive architecture, while it interacts with the Isis World physical simulation.

Part III

CONCLUSION

6

RESULTS

7

DISCUSSION

During our initial experiments, we discovered many organizational benefits of representing parts of our system in a reflectively traced form.

7.1 REFLECTIVE KNOWLEDGE MAINTAINANCE

7.2 SCRIPT DECOMPOSITION AND RECOMPOSITION

8

FUTURE

8.1 PANALOGY ARCHITECTURE

8.1.1 *Recursive Loops and Infinite Recursive Tracing Descent*

If the focus on the tracing is controlled carefully, these potential loops can be avoided. How to detect and control these potential loops is an interesting area of future automatic debugging research in reflective control.

8.1.2 *Potential Future Uses for Low-Level Tracing*

Lower level objects maybe be interesting to focus on for research in automatic abstraction and simulation of system components. Optimizing compilers could benefit from this area of future research. E.g. focusing on the CPU object could help to develop better run-time register allocation models.

8.1.3 *Why Should You Use This Radically New Language?*

Because the language is very similar to Lisp, it has proven to be easy for both expert and novice programmers to learn. This has been my experience with the four undergraduates that have worked within the language, who learned it quickly, started writing their own macros to facilitate their style, and one even made additions to the core algorithms.

8.2 AGENT SPEECH ACTS

At any given point in time, each agent is pursuing a different variety of goals. Each agent executes actions in order to accomplish collections of these goals. We treat language communication as an additional physical action that the agent can choose to perform in order to accomplish its goals.

8.3 MODELLING NOISE IN AGENT COMMUNICATION

Our cognitive theory makes no statements as to the presence of noise in communication channels between agents and the environment. Communication between agents and the environment in our implementation occurs across a noiseless communication channel. It is theoretically possible to include a theory of noisy communication channels between agents and the environment. For example, Shannon's mathmematical theory of communica-

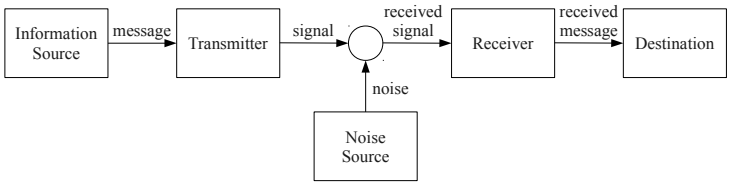


Figure 29: A mathamtical theory of communication (Shannon, 1959).

tion (Shannon, 1959) could be applied as an elaboration to any of the information pathways in our basic theory.

Part IV

APPENDIX

A

RELATED PHILOSOPHY

A.1 THE OBJECTIVE MODELLING ASSUMPTION

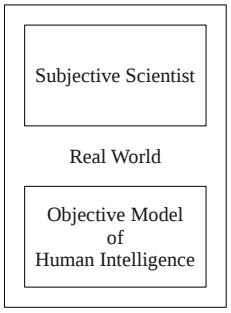


Figure 30: The objective-subjective modelling assumption.

We assume that the phenomenon that we are trying to model, namely human intelligence, is an objective process that we can describe. This is the objective-subjective philosophical assumption that is inherent in any objective scientific hypothesis. We make this assumption in order to avoid logical problems of circular causality that occur when trying to find a non-objective description of reflective thinking. Figure 30 shows how, given the objective assumption, the subjective scientist is part of the real world, while she is studying an objective phenomenon. Given the objective-subjective assumption, it would be a grave mistake to confuse an objective model for reality itself.

A.2 BEING, TIME, AND THE VERB-GERUND RELATIONSHIP

A.3 THE INTENSIONAL STANCE

A.4 REFLECTIVE REPRESENTATIONS

(Perner, 1991)

B

RELATED PSYCHOLOGY

Between the ages of 1-3 years old, children display primary emotions, such as joy, disappointment, and surprise. These emotional processes have been hypothesized to be related to the process of failing or succeeding to accomplish a goal. Around age 4, children begin to display emotions that involve the self, such as guilt and shame. It has been hypothesized that these emotions relate to another person's evaluation of the child's goals as good or bad.

We approach modelling this developmental process by applying Marvin Minsky's theory of the child-imprimer relationship. According to Minsky's theory, at a young age, a human child becomes attached to a person that functions as a teacher. The imprimer could be a parent or a caregiver or another person in the child's life, but the function of the imprimer is to provide feedback to the child in terms of what goals are good or bad for the child to pursue.

B.1 SIMULATION THEORY OF MIND VERSUS THEORY THEORY OF MIND

B.2 EMOTION OR AFFECT VERSUS GOAL-ORIENTED COGNITION

B.3 EMBARRASSMENT, GUILT, AND SHAME

C

RELATED NEUROSCIENCE

D

RELATED ARTIFICIAL INTELLIGENCE

D.1 PLANNING ASSUMING A CORRECT MODEL OF ENVIRONMENT

There are many types of processes that make plans for accomplishing goals, which make plans assuming a model of how actions theoretically affect the environment. These processes are called planners when they create a representation for how actions should be performed, potentially including temporal ordering constraints.

Planners are a specific part of a complete learning system, but the primary function of a planner is to find a theoretical solution to a given problem. This theoretical solution, or plan, may be executed and may fail or succeed, in accordance with the initial intentions for executing the plan, the initial intentions for imagining the plan, or any other intentions. If the plan fails, then we may find something to be modified in our knowledge in order to help us in avoiding this failure next time. The planning process is a small part of the complete closed-loop learning algorithm that learns to plan from experience with the environment and other agents.

If we are thinking about the temporal constraints of the problem solving process itself, then we need to consider a reflective approach to this control problem.

, (1) the model of the cause-effect relationship between actions and the world, (2) the model of cause-effect relationships between planning actions and the creation of successful plans.

D.2 DECLARATIVE PROGRAMMING, LOGICAL REASONING

D.3 WHY DID I FORGET TO INCLUDE PROBABILITY IN MY THEORY?

D.4 THE REINFORCEMENT LEARNING MODEL

Figure 31 shows the basic reinforcement learning model. This model is an agent environment model, but there is an extra information channel from the environment to the agent, which communicates a numerical reward signal. We can now say that the agent has a learning problem. The agent must learn what actions to execute in order to gather the most reward.

Once we have a basic reinforcement learning algorithm, we can approach this learning problem as a function approximation problem. In other words, we can try to learn what parts of the perception and action space have more or less reward. Figure 32

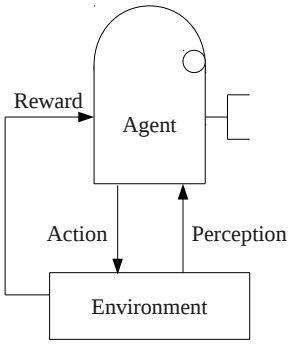


Figure 31: The reinforcement learning model.

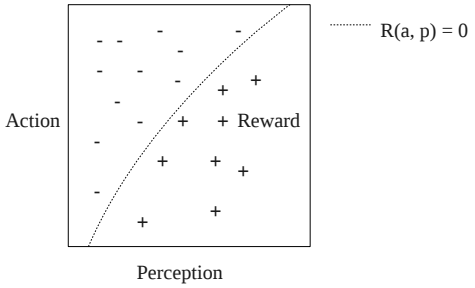


Figure 32: Categorizing perceptions and actions based on goals.

shows a diagram of this state space with the zero crossing of an approximation of the reward plotted.

D.4.1 *Finding a Good Policy for Gathering Rewards*

Learning an approximation of what parts of a state space are good or bad, based on reward, is not all that is needed to determine what actions the agent should perform. The agent wants to gather the most rewards over time. A simple way to formalize this problem is to learn a policy that determines what action should be executed for every part of the state space, based on some sort of summation of rewards over time. There have a been a number of ways of formalizing this summation process as finite or infinite horizon problems (Sutton & Barto, 1998). Dynamic programming can be used for finding an optimal or an approximately optimal policy (Bertsekas, 1995).

D.4.2 *Categorizing Perceptions and Actions based on Goals*

One problem with the reinforcement learning approach is that the only representation of success or failure is a single number, the reward. The basic reinforcement learning problem has been defined for finite propositional state spaces.

A representation called Relational Markov Decision Process (RMDP) has been proposed (Guestrin *et al.* , 2003) in order to extend reinforcement learning to larger relational problem domains, but this method only focuses on an object-oriented reward that

does not have any global feedback about the overall value of the situation.

E

RELATED COMPUTER SCIENCE

E.1 LADDER FINITE STATE MACHINE

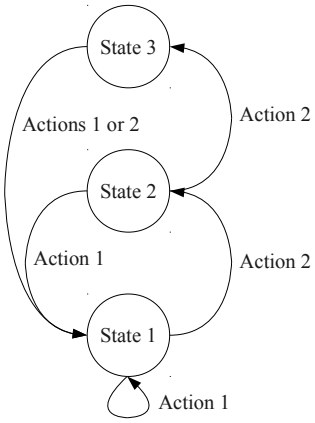


Figure 33: Ladder state machine with three states and two actions for each state.

See Figure 33.

E.2 CLOUD COMPUTING

E.3 DATABASES AND KNOWLEDGE REPRESENTATION

F

APPLICATIONS TO MENTAL HEALTH

G

APPLICATIONS TO EDUCATION

H

THE CODE

H.1 OPEN-SOURCE DOWNLOAD

Everything is open-source, can be downloaded from my webpage, and compiled by simply typing "./configure; make".

H.2 THE HACKER PHILOSOPHY OF CODE

The problems of artificial intelligence and greater control of more complicated and interconnected computational systems are not easy domains to approach from a programming perspective. Many of these problems seem to require not only new programming abstractions, methodologies and philosophies, but also bug-free implementations of these potentially paradigm shifting ideas.

In other words, the only people that are going to be able to solve these types of control problems that are on the forefront of science fiction in research are going to be expert computer programmers, or "hackers." In general the ideal hacker is a problem solver that accomplishes their own goals in given complicated systems, not necessarily computer systems.

In terms of computational science, hackers are often confronted by given hardware and software problems that must be overcome in order to implement solutions to their posed problems. Hacking can be adopted as a fun philosophy for accomplishing ones own goals in life, but also, I think that hacking is a required prerequisite for any form of scientific progress. A joy for tinkering that leads to playing, which subsequently leaves the tinkerer as an expert hacker is what is needed for approaching undocumented domains, such as the forefront of artificial intelligence and control theory in general.

H.3 WHAT IS A COMPUTER?

Good question. We've presented a simple model in Section 1.1.1, but this abstract model leaves a lot to be desired for modelling the more realistic computers that most programmers enjoy. For example, hardware platforms with many processing cores per CPU, multiple CPUs per motherboard, and multiple computers per networked cluster of motherboards. In order to reflectively control computational systems that are organized according to these modern engineering constraints, I have organized my memory infrastructure to span these areas of future reflective control research. For example, basic pointer in the Funk2 programming language includes 17 bits to represent the cluster machine ID, and

9 bits to represent a processor core specific memory allocation pool. We see Funk2 as a platform supporting an open research community of hackers that share computational resources in order to build demonstrations of larger reflective artificial intelligence control systems.

BIBLIOGRAPHY

Bertsekas, D.P. 1995. Dynamic programming and optimal control.

Bringhurst, Robert. 2002. *The Elements of Typographic Style*. Version 2.5. Point Roberts, WA, USA: Hartley & Marks, Publishers.

Cox, Michael T. 2005. Metacognition in computation: A selected research review. *Artificial Intelligence*, **169**(2), 104 – 141. Special Review Issue.

Dewey, John. 1907. *The School and Society: The School and the Life of the Child*. Chicago: University of Chicago Press. Chap. 2, pages 47–73.

Guestrin, C., Koller, D., Gearhart, C., & Kanodia, N. 2003. Generalizing plans to new environments in relational MDPs. *In: In International Joint Conference on Artificial Intelligence (IJCAI-03*. Citeseer.

Hume, David. 1902. *Enquiries concerning the human understanding: and concerning the principles of morals*. Vol. 921. Clarendon Press.

Lenat, D.B., Guha, R.V., Pittman, K., Pratt, D., & Shepherd, M. 1990. Cyc: toward programs with common sense. *Communications of the ACM*, **33**(8), 30–49.

Liu, H., & Singh, P. 2004a. Commonsense reasoning in and over natural language. *Pages 293–306 of: Knowledge-Based Intelligent Information and Engineering Systems*. Springer.

Liu, H., & Singh, P. 2004b. ConceptNet—a practical commonsense reasoning tool-kit. *BT technology journal*, **22**(4), 211–226.

Matuszek, C., Witbrock, M., Kahlert, R.C., Cabral, J., Schneider, D., Shah, P., & Lenat, D. 2005. Searching for common sense: Populating Cyc from the Web. *Page 1430 of: Proceedings of the National Conference on Artificial Intelligence*, vol. 20. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

McCarthy, J. 1994. Elephant 2000: A programming language based on speech acts. *Unpublished Manuscript, Stanford University*.

McDermott, D.V. 1987. Logic, problem solving, and deduction. *Annual Review of Computer Science*, **2**(1), 187–229.

Minsky, Marvin. 1988. *The society of mind*. Simon and Schuster.

Minsky, Marvin. 1991. Logical vs. analogical or symbolic vs. connectionist or neat vs. scruffy. *Pages 218–243 of: Artificial intelligence at MIT expanding frontiers*. MIT press.

Morgan, Bo. 2009. Funk2: A Distributed Processing Language for Reflective Tracing of a Large Critic-Selector Cognitive Architecture. *Proceedings of the Metacognition Workshop at the Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*.

Morgan, Bo. 2010. A Computational Theory of the Communication of Problem Solving Knowledge between Parents and Children. *PhD Proposal*, January.

Morgan, Bo. 2011. Moral Compass: Commonsense Social Reasoning Cognitive Architecture. *Commonsense Tech Note*, January.

Newell, Alan, Shaw, Cliff, & Simon, Herbert. 1959. Report on a general problem-solving program. *Pages 256–264 of: Proceedings of the International Conference on Information Processing*.

Newell, Allen. 1990. *Unified theories of cognition*.

Orkin, J., & Roy, D. 2009. Automatic learning and generation of social behavior from collective human gameplay. *Pages 385–392 of: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems.

Orkin, J., Smith, T., Reckman, H., & Roy, D. 2010. Semi-automatic task recognition for interactive narratives with EAT & RUN. *Pages 1–8 of: Proceedings of the Intelligent Narrative Technologies III Workshop*. ACM.

Perner, Josef. 1991. *Understanding the representational mind*. MIT Press.

Shannon, C.E. 1959. A mathematical theory of communication. *The Bell System Technical Journal*, **27**(July, October), 379–423, 623–656.

Simon, H.A. 1972. Theories of bounded rationality. *Decision and organization*, **1**, 161–176.

Singh, P., Lin, T., Mueller, E., Lim, G., Perkins, T., & Li Zhu, W. 2002. Open Mind Common Sense: Knowledge acquisition from the general public. *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, 1223–1237.

Singh, Pushpinder. 2005 (June). *EM-ONE: An Architecture for Reflective Commonsense Thinking*. Ph.D. thesis, Massachusetts Institute of Technology.

Smith, Dustin, & Morgan, Bo. 2010. IsisWorld: An open source commonsense simulator for AI researchers. *In: Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Speer, R., Havasi, C., & Lieberman, H. 2008. AnalogySpace: Reducing the dimensionality of common sense knowledge. *In: Proceedings of AAAI*.

Stein, G., & Barnden, J.A. 1995. Towards more flexible and common-sensical reasoning about beliefs. *Pages 127–135 of: Proceedings of the 1995 AAAI Spring Symposium on Representing Mental States and Mechanisms*.

Sutton, R.S., & Barto, A.G. 1998. *Reinforcement learning*. Vol. 9. MIT Press.

Winograd, Terry. 1970. *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language*. Ph.D. thesis, Massachusetts Institute of Technology.

COLOPHON

This thesis was typeset with LaTeX 2ε using Hermann Zapf's *Palatino* and *Euler* type faces (Type 1 PostScript fonts *URW Palladio L* and *FPL* were used). The listings are typeset in *Bera Mono*, originally developed by Bitstream, Inc. as "Bitstream Vera". (Type 1 PostScript fonts were made available by Malte Rosenau and Ulrich Dirr.)

The typographic style was inspired by Bringhurst's genius as presented in *The Elements of Typographic Style* (Bringhurst, 2002). It is available for LaTeX via CTAN as "`classicthesis`".

NOTE: The custom size of the textblock was calculated using the directions given by Mr. Bringhurst (pages 26–29 and 175/176). 10 pt Palatino needs 133.21 pt for the string "abcdefghijklmnopqrstuvwxyz". This yields a good line length between 24–26 pc (288–312 pt). Using a "*double square textblock*" with a 1:2 ratio this results in a textblock of 312:624 pt (which includes the headline in this design). A good alternative would be the "*golden section textblock*" with a ratio of 1:1.62, here 312:505.44 pt. For comparison, `DIV9` of the `typearea` package results in a line length of 389 pt (32.4 pc), which is by far too long. However, this information will only be of interest for hardcore pseudo-typographers like me.

To make your own calculations, use the following commands and look up the corresponding lengths in the book:

```
\settowidth{\abcd}{abcdefghijklmnopqrstuvwxyz}
\the\abcd\ % prints the value of the length
```

Please see the file `classicthesis.sty` for some precalculated values for Palatino and Minion.

145.86469pt

DECLARATION

I, Bo Morgan, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Cambridge, August 2011

Bo Morgan