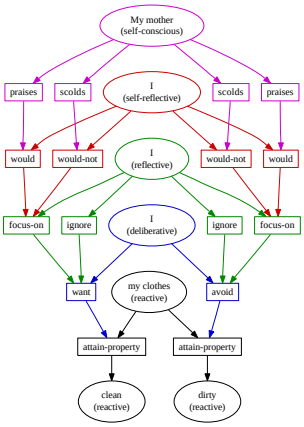
BO MORGAN

A SUBSTRATE FOR ACCOUNTABLE LAYERED SYSTEMS

A SUBSTRATE FOR ACCOUNTABLE LAYERED SYSTEMS

BO MORGAN



Ph.D. in the Media Arts and Sciences

August 2011

"If wishes were horses, beggars would ride." Since they are not,
since really to satisfy an impulse or interest means to work it out,
and working it out involves running up against obstacles,
becoming acquainted with materials, exercising ingenuity,
patience, persistence, alertness, it of necessity involves
discipline—ordering of power and supplies knowledge.
— John Dewey

Don't do anything that isn't play.
— Joseph Campbell

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 cognitive architecture simulation of life in a rigidbody physical environment. 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:

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.; "Funk2: A Frame-based Programming Language with Causally Reflective Capabilities (draft in progress)"; Technical Note; Massachusetts Institute of Technology; 2009 May

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

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

ACKNOWLEDGMENTS

Put your 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

THE PROBLEM

1

MODELS OF COGNITION INTRODUCTION

> 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 first chapter I will give an overview of the problem of social problem solving. I will begin with a model of a single problem solver in an environment. Discuss the idea of a simple numerical goal for the agent and then extend this model to a relational domain with abstractions over states. The goal-oriented learning problem will be introduced, and then extended to a multiple agent problem solving model, where each agent may be pursuing different goals. I will use a simple kitchen cooking scenario to motivate our model of learning to solve problems in a social agent environment.

In later chapters I will introduce experiments I have performed with a closed-loop system that learns—not simply from a one dimensional reward signal, but from inheriting cultural knowledge from other agents and reflectively debugging the use of this knowledge when it has been used by the planner and fails in execution. One type of information that my model transmits between agents is similar to plan representations that are used in the planning community, such as sequences of goal states to be achieved. These high-level procedures are specified in a language that is shared by the agents, and refers to both mental and physical actions of the agents.

The social problem solving agent is interesting because of the lack of knowledge that one agent has of the other agents' minds—their goals and beliefs.

1.1 THE AGENT ENVIRONMENT MODEL

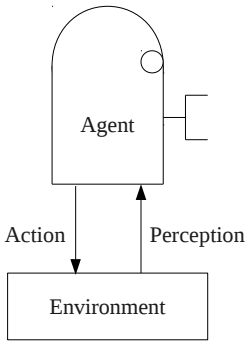


Figure 1: The agent environment model.

Figure 1 shows the basic agent environment model. In this model, we 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. We 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.

1.2 THE STATE-ACTION MODEL

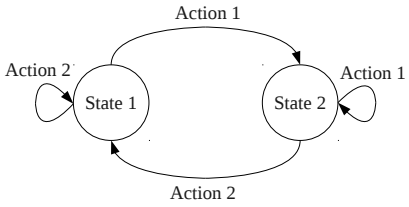


Figure 2: The state action model. Two states are represented by nodes and two actions are represented by edges from each of the two states.

The agent is in an environment, which is in a specific state. Our 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.

1.3 RELATIONAL REPRESENTATIONS AS AN ABSTRACTION OVER STATE SPACES

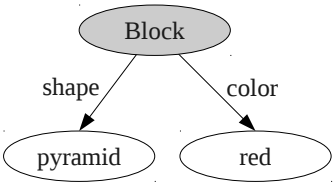


Figure 3: Frame-based relational representation.

See Figure 3.

1.4 FRAME PERCEPTIONS

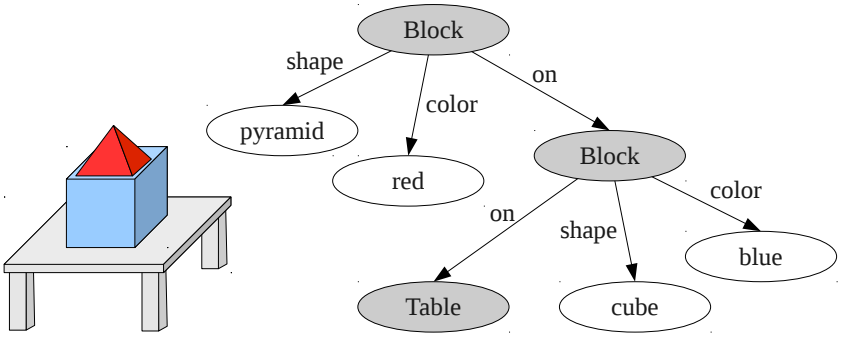


Figure 4: Collections of frames used as perceptual input to agent.

See Figure 4.

1.5 PARTIALLY OBSERVABLE STATE MODEL

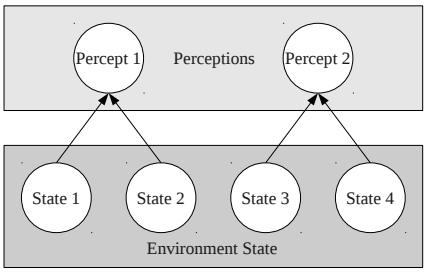


Figure 5: The partially observable model.

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.

1.6 PARTIAL FRAME PERCEPTIONS

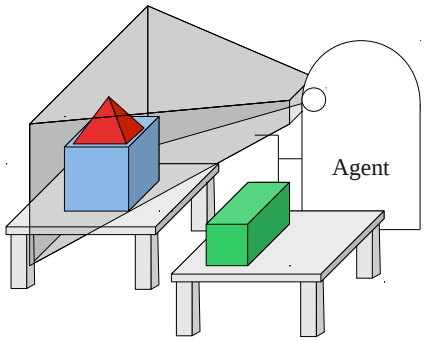


Figure 6: The state of the environment is only partially observable.

See Figure 6.

1.7 AGENT ABSTRACT PHYSICAL MODEL

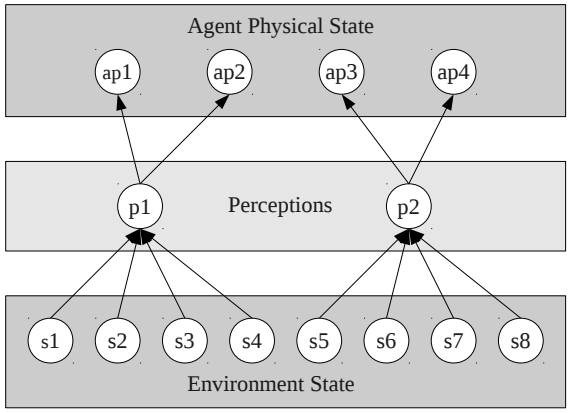


Figure 7: The agent has an abstract physical model of the environment.

See Figure 7.

1.8 PERCEPTUAL SUPPORT OF PHYSICAL KNOWLEDGE

See Figure 8.

1.9 A PHYSICAL GOAL REPRESENTATION

See Figure 9.

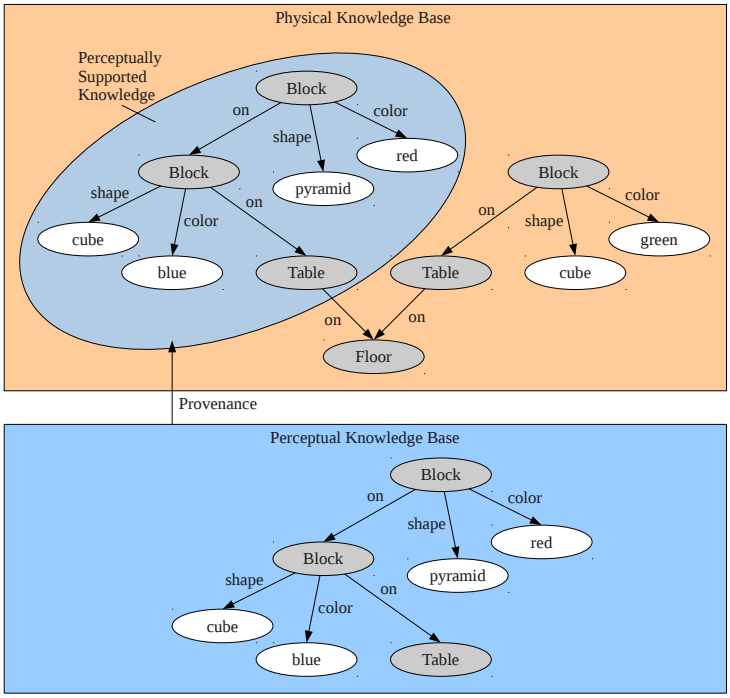


Figure 8: Perceptual provenance provides support for physical knowledge.

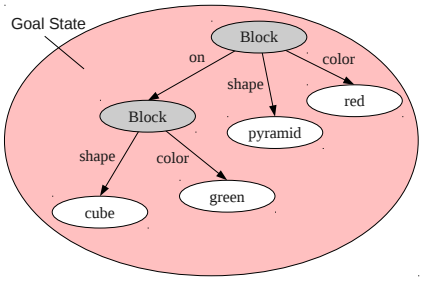


Figure 9: A physical goal representation is a structural relationship that may or may not currently exist within the physical knowledge base.

1.10 THE MULTIPLE AGENT ENVIRONMENT MODEL

In order to model social intelligence, we introduce the multiple agent environment model shown in Figure 10. One interesting property of the multiple agent environment model is that any communication between any two agent processes must occur through the environment process.

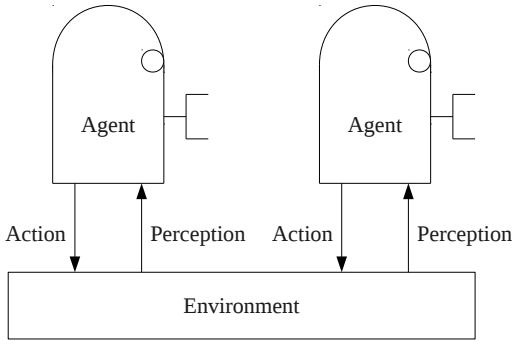


Figure 10: The multiple agent environment model.

2

LITERATURE OF COGNITION AND COMMONSENSE

2.1 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.

2.1.1 *Adaptability in Complex Environments*

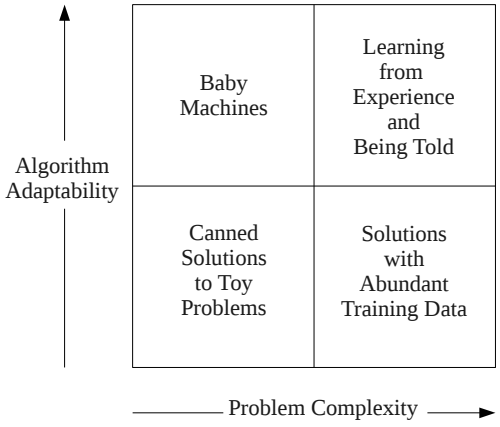


Figure 11: Problem complexity versus algorithm adaptability.

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 algorithms 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 11 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.

2.1.2 *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).

2.2 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.

2.2.1 *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

2.3 COMPARABLE COGNITIVE ARCHITECTURES

EM-ONE, Cyc, Icarus, ACT-r, Soar, and Prodigy are cognitive architectures.

3

PROBLEMS TO SOLVE

4

THEORY AND ALTERNATIVES

4.1 FEEDBACK CONTROL MODEL FOR ACCOMPLISHING A SINGLE GOAL

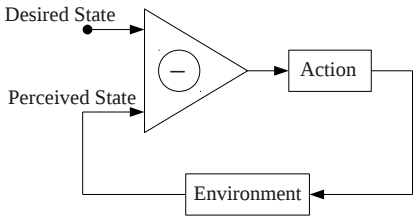


Figure 12: The feedback control model for accomplishing a single goal.

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 example, Figure 12 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.

4.2 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.

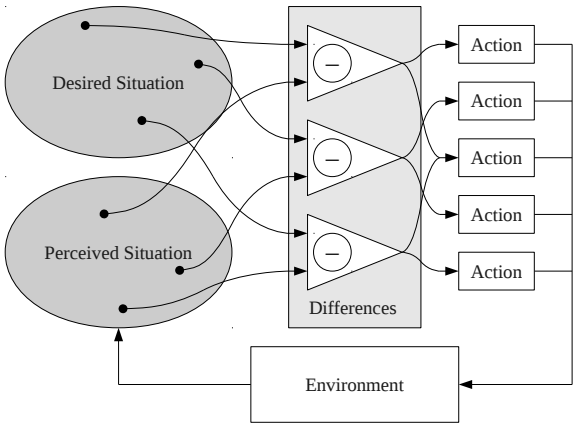


Figure 13: The difference engine model for accomplishing multiple goals.

4.3 DIFFERENCE-ENGINE MODEL FOR ACCOMPLISHING MULTIPLE GOALS

(Minsky, 1988, p. 78)

Part II

OUR SOLUTION

5

A SYSTEM

5.1 REFLECTIVE KNOWLEDGE

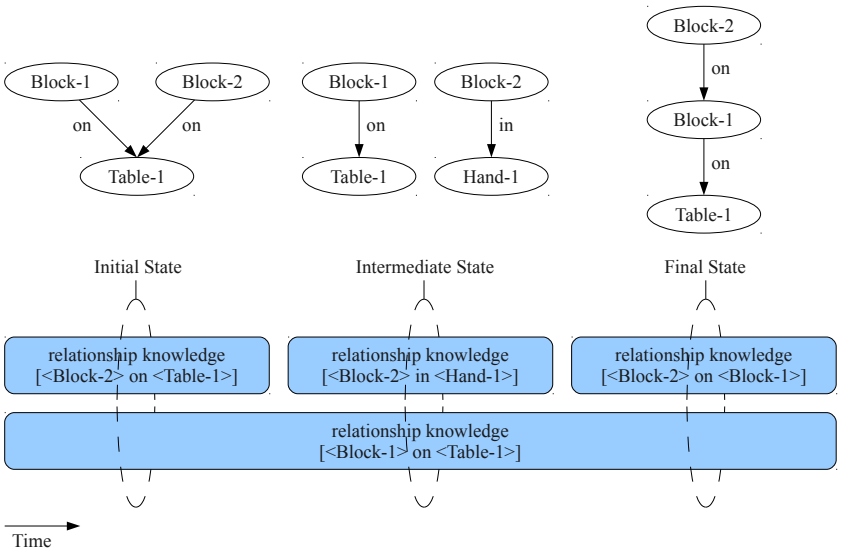


Figure 14: A reflective event representation shows the changes in a labeled graph.

While the term meta-knowledge is used to describe the very general idea of knowledge about knowledge, we 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 we keep track of the changes to a knowledge structure, we 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.

5.2 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. We will discuss the various methods for controlling the tracing focus of the reflective memory that we have found useful in the development of our system in Section 5.6.

5.3 DEBUGGING PLANS BY REFLECTIVELY TRACING THE PROVENANCE OF KNOWLEDGE

The thesis is focusing on provenance of knowledge. From perception to other knowledge representations, through planning and failed steps. Normally, when plans fail, rule learning (or other learning method) is used to update beginning and ending conditions for actions. Very complex planning methods exist that all assume that the current world model is perfect (in either a deterministic or probabilistic representation of the world). Currently, plans are called policies that handle all possible contingencies, in which the agent may find itself. 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 in a hill-climbing sense. Rethinking this problem with my new approach that maintains provenance for knowledge used in creating plans allows debugging the complex web of knowledge and processes involved in creating that knowledge—rather than simply updating the rules related to a single action.

The planner that I've built in my system is not as complicated as state of the art planners that currently exist in the field, but my planner allows a reflective process to debug the failure of plans created by our planner by tracing the provenance of the knowledge in the plan representation itself.

5.4 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. We see the problem as requiring a combination of many different types of representations and reasoning processes.

We worked with Pushpinder Singh from 1999 to 2006 on the first version of the Emotion Machine architecture, EM1 (Singh, 2005). During that period, we discussed that one weakness in the EM1 system is its reliance on tracing only the declarative prolog statements, among other necessary but untraced procedural code. Although EM1 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.

Because the Emotion Machine theory of mind requires a variety of many reasoning processes to be controlled, we have built an operating system on top of this traceable memory layer.

Further, the Emotion Machine is organized into layers of different representations, so we have built a lisp-like language on top this operating system. The benefit of having a lisp-like language is the language's ability to efficiently represent and simulate new programming languages.

Within our lisp-like language we have written an implementation of a layered reflective cognitive architecture, inspired by EM1.

Finally, we demonstrate our cognitive architecture in a rigid-body physical environment, where multiple agents demonstrate learning from both being told solutions to problems as well as learning from the environment in which situations these solutions succeed or fail. We show how our 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.

5.5 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 (reference needed; I think you mentioned this in class), 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, we 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 we need to make

distinctions between situations that appear identical. If we need to learn a new rule for categorization of situations, or even a new category entirely, we need to go back to the correct features and processes that performed those categorizations of the identical situations that we need to further distinguish.

5.6 METHODS FOR FOCUSING REFLECTIVE TRACING

Section 5.6 is referenced from Section 5.2.

5.7 AN OPERATING SYSTEM

We've written a multi-core operating system, including a compiler, on top of this traceable and distributed memory layer.

Because the system is meant to combine many artificial intelligence techniques, we have tested our platform running thousands of parallel processes concurrently on an 8-core machine. These processes can control and watch one another execute. We 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 we strongly disagree. We need good software engineers in order to solve most of the problems we face in getting these massive software systems to work together.

5.8 A PROGRAMMING LANGUAGE

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

5.8.1 *Why not use Lisp?*

Lisp is a great programming language. We 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. We 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 we 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 we found ourselves 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.

5.9 A LAYERED COGNITIVE ARCHITECTURE

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

5.10 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.

5.10.1 *Trace Only an Appropriate Level of Abstraction*

It does not make sense to trace below a certain depth of processing. If we 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.

5.10.2 *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).

6 EXPERIMENTS

6.1 A DEMONSTRATION IN A SOCIAL COMMONSENSE REASONING DOMAIN

The goal of our project is to build a demo of part of Minsky's architecture, and the domain of commonsense reasoning in a kitchen is I feel the right way to develop those high-level theories.

We tested our cognitive architecture learning in the context of a social commonsense reasoning domain with parents that teach children as they attempt to accomplish cooking tasks in a kitchen. Kitchens are a good example of a rich learning environment for children (Dewey, 1907). Kitchens are ubiquitous across cultures. They have a clear production goal, food. They involve many many mental realms: math, physics, chemistry, thermodynamics, natural language, social, family, imprimer learning, children, parents, concurrent planning, etc.

6.2 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 we do not ignore this aspect in tracing the deliberative layer. Our 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 know not only know the ending point of computation but also the means toward that end.

6.2.1 *Why Not Work Within a Building Blocks Domain?*

The building blocks approach is a good precedent. I did not want to get buried in the morass of common sense knowledge problems about cooking, but I did want to approach a domain in which there are a variety of objects requiring complex models of cause and effect.

We have conscripted our 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. Our object types do have different ways that they may be used, which is a small addition of complexity. Although we do not introduce many of the complexities of ontological reasoning, a common approach to

commonsense reasoning, e.g. Cyc (Lenat *et al.* , 1990), our 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.

6.2.2 *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.

Part III

CONCLUSION

7

DISCUSSION

8

FUTURE

8.0.3 *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.

8.0.4 *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.

8.0.5 *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.

Part IV

APPENDIX

A

RELATED PHILOSOPHY

A.1 THE OBJECTIVE MODELLING ASSUMPTION

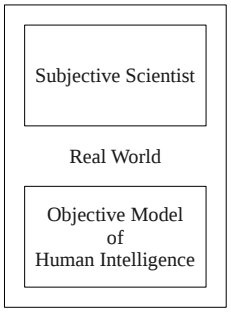


Figure 15: The objective-subjective modelling assumption.

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 15 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 THE REINFORCEMENT LEARNING MODEL

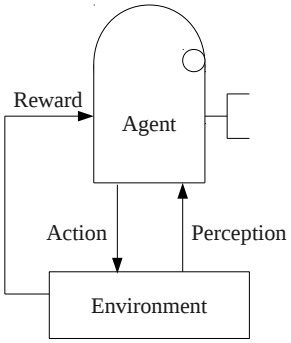


Figure 16: The reinforcement learning model.

Figure 16 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.

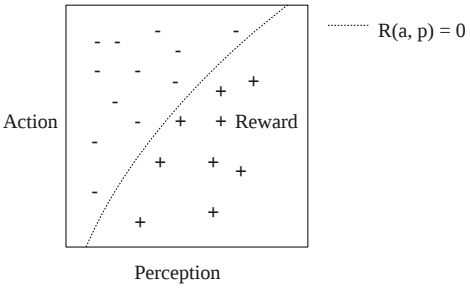


Figure 17: Categorizing perceptions and actions based on goals.

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

D.1.1 *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.1.2 *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

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".

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.

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.

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.

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.

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

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

Put your declaration here.

Cambridge, August 2011

Bo Morgan