BO MORGAN

A SUBSTRATE FOR ACCOUNTABLE LAYERED SYSTEMS

Advisor ____________________
Joseph A. Paradiso
Professor of Media Arts and Sciences
Massachusetts Institute of Technology

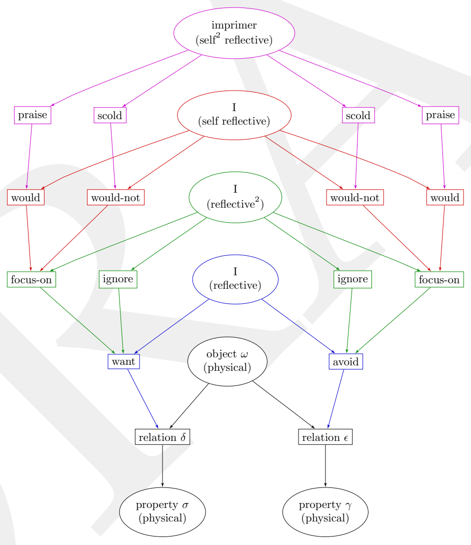
Committee Member
____
Marvin Minsky
Professor of Media Arts and Sciences
Professor of Electrical Engineering and Computer Science
Massachusetts Institute of Technology

Committee Member
____
Gerald J. Sussman
Panasonic Professor of Electrical Engineering
Massachusetts Institute of Technology

Committee Member
____
Michael T. Cox
Visiting Associate Research Scientist
University of Maryland

A SUBSTRATE FOR ACCOUNTABLE LAYERED SYSTEMS

BO MORGAN



Ph.D. in the Media Arts and Sciences

Media Lab
Massachusetts Institute of Technology

August 2012

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.

— Hume (1902)

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.

This constructed substrate 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, making 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 problem domain. In my demonstration an AI model can learn from experience of success or failure. The AI not only learns about physical activities but also reflectively learns about thinking activities, refining and learning the utility of built-in knowledge. Procedurally traced memory can be used to assign credit to those thinking 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 would first like to thank Push Singh for being a strong memory of a heroic friend and advisor.

I would like to thank my committee: Joe Paradiso for his belief in my ability, Marvin Minsky for consistently providing critical perspective and important areas for research, Gerry Sussman for teaching a love of programming, and Mike Cox for leading in the space of metacognitive AI.

I would like to thank my immediate family: Greg Morgan and Carolyn Spinner for being willing to help me think about anything at any time. Paul Bergman for teaching me how to program. Virginia Barasch for making it a party. Leaf Morgan for playing Archon.

I'd like to thank those who worked on the project: Dustin Smith for the physical social commonsense kitchen simulation. Radu Raduta for the PID control loops for robot balance and movement. Jon Spaulding for the EEG amplifiers and pasteless electrodes. Gleb Kuznetsov for the first version of a 3D commonsense kitchen critic-selector layout planner. Jing Jian for being so good at finding critical bugs in Funk2 and for writing all of the learned-reactive resources for following recipes in the kitchen. Panupong Pasupat for the error-correcting subgraph isomorphism algorithm and the Winston difference-of-difference analogy algorithm. Mika Braginsky for the visualization of mental resources, agencies, and layers.

I'd like to thank the following professionals for their encouragement, advice, and inspiration: Walter Bender, Henry Lieberman, Patrick Winston, Whitman Richards, Aaron Sloman, Ed Boyden, Sebastian Seung, Ted Selker, Hugh Herr, and Rebecca Saxe.

I'd like to thank my friends in the Common Sense Computing group: Catherine Havasi, Rob Speer, Hugo Liu, Barbara Barry, Ian Eslick, Jason Alonso, and Dustin Smith.

I would like to thank my popes of GOAM: Scotty Vercoe, Dane Scalise, Kemp Harris, and Dustin Smith.

I would like to thank the researchers of the Mind Machine Project for being an inspiring shooting gallery of high-energy particle AI: Newton Howard, Forrest Green, Kenneth Arnold, Dustin Smith, Catherine Havasi, Scott Greenwald, Rob Speer, Peter Schmidt-Nielsen, Sue Felshin, David Dalrymple, and Jason Alonso.

I would be remiss if I were not to thank, again in some cases, the following individuals, whom have kept me laughing through the PhD process: Dustin Smith, Grant Kristofek, Mako Hill, Josh Lifton, Mat Laibowitz, Mark Feldmeier, Matt Aldrich, Nan-Wei Gong, Mike Lapinski, Nan Zhao, Clio Andris, Hannah Perner-Wilson, Jeff Lieberman, Edwina Portocarrero, David Cranor, and Emöke-Ágnes Horvát.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

In this dissertation I present the substrate for accountable layered systems. I have focused on building this substrate for doing reflective thinking on a large scale in learning systems, using Singh's (2005) example of reflective architectures as the precedent. My approach focuses on a purely procedural approach that does not assume any logical search algorithms that work behind-the-scenes without a reflective learning focus. Without increasing the computational time complexity of basic search, this approach learns to search in n concurrent layers of goal-oriented optimization, learning to plan physical actions, while concurrently learning to plan planning actions, etc. Physical activities are presented as analogous to thinking activities, allowing a recursive application of the model to itself. The contributions of this thesis are:

- *The Model*: A philosophical basis for avoiding tautologies while thinking about a grounded model of reflection.
- *The Simulation*: A mathematical basis in graph theory for representing a simulation of the model.
- *The Substrate*: A computational implementation of three working layers of the simulation in a block building domain.

1.1 THE SUBSTRATE

The primary contribution is the substrate. This is a system for developing large software applications, which enables parallel and concurrent process control. The advantage is the system's ability to selectively pause these parallel processes when bugs are encountered, so that they can be easily examined in this paused state. Additionally, there are intricate thread control operations for these processes. When they "sleep", they are removed from the scheduler, allowing the substrate to focus computational resources on other processes. When required, thousands of sleeping processes can efficiently be "triggered" awake. The primary point of the system is that while running many different problem solvers at the same time, it has the ability to conveniently trace the interactions between those problem solvers. Thus, all of the memory in this substrate allows for the tracing of all memory allocations and mutations. It is a frame-based system. Many AI systems are implemented in an object-oriented or frame-based representation. This allows for all of the commonly associated programming methodologies to exist, i.e. object types and inheritance, etc. The substrate includes a high-level lisp-like programming language, so the compiler can be redefined by the running program. There are layered "cognitive architecture" structural primitives included in the language. This allows the programmer to build a "layer" which contains "agencies" which contain "resources". These resources then are able to exist as "minds" which control "physical worlds" because these primitives already exist in this substrate. This substrate, which can be downloaded as open-source software, serves as an extensible platform for continuing research on this implementation of reflective thinking.

1.2 THE SIMULATION

The second thesis contribution is an example of a mathematical simulation model that gives answers for the following questions:

- How do we assign credit to the planning process and learn to plan?
- When many individual processes interact in the solution of a common problem, for example, controlling a complicated parallel system, how is credit traced for decisions that have been made in performing actions?
- How do we learn what particular knowledge is either good or bad at solving different types of problems?
- How do we trace dependencies back to the point that models of planning can be learned?
- How do algorithms currently assign credit backwards in time and how does this model relate to these methods?

In this thesis, I evaluate how well this model is able to learn, not only at the ground level but also at the hypothesis manipulation level, improving the ground learning performance. This simulation will be described as a potential path around the exponential search explosion problem. There is a certain "curse of dimensionality," which limits a search algorithm from considering an exponentially vanishing percentage of the possible search paths as the length of the paths is increased. The simulations are often defined in abstract terms, such as logical relationships, in order to make abstract simulations of large domains. Large systems that have logical relationships require search algorithms to evaluate the implications of the declarative forms. These systems fail because there is this explosion of exponential search. In my model, in analyzing the simulation, the curse of dimensionality is reduced through layers of planned heuristic learning. My main point is that anything that involves search can be improved by being in terms of a simulation of reflective learning. The details of the simulation model are the basis of the computational substrate that allows for this improvement.

1.3 THE MODEL

The third thesis contribution is a reflective model of mind that gives a description of how to think about thinking reflectively. There are many models for reflective thinking, and I will compare my model to other related layered, reflective, and metacognitive models of mind. The model I describe here has not been explored by many of the current machine learning disciplines. Thus, this model will be the basis of my mathematical simulation.

1.4 DOCUMENT OVERVIEW

Part I begins with a non-technical description of the model of mind that is used as the basis of both the mathematical simulation and the computational implementations. In Part II, I explain the modelling assumptions that I make in transitioning to the mathematical notation of the simulation model. Using this notation, I explain how this model can be used to reduce the complexity of search algorithms. In Part III, I give an explanation of how this simulation is automated on a concurrent computer, the thesis implementation, SALS, the substrate for accountable layered systems. In conclusion, in Part IV, I discuss promising directions for future research in not only AI but also the other cognitive sciences.

The focus of the dissertation will be a description of the reflective problem of learning-to-control. I describe control in terms of layers of reflective credit assignment because this simplifies understanding the problem of learning-to-control. The focus will be on the implementation of credit assignment at the level of planning activities and then describe an example of a computational simulations of reflective learning in a block building physical domain.

1.5 PROGRAMMING

An example problem domain called the block building domain is shown in Figure 1. In this case, we have two blocks, which have relationships with the objects in the room, such as the table. If the goal state is to get the physical world to exist in a particular configuration, the stack of `Block-2` on top of `Block-1`, Figure 2 shows an example of a program that a programmer might write to move this gripper around in this block building domain. In this program, the gripper has very simple symbolic commands: `move-right`, `reach`, `grab`, `move-left`, `drop`. One can imagine that this program might intend to pick up the green block and put it on the blue block.

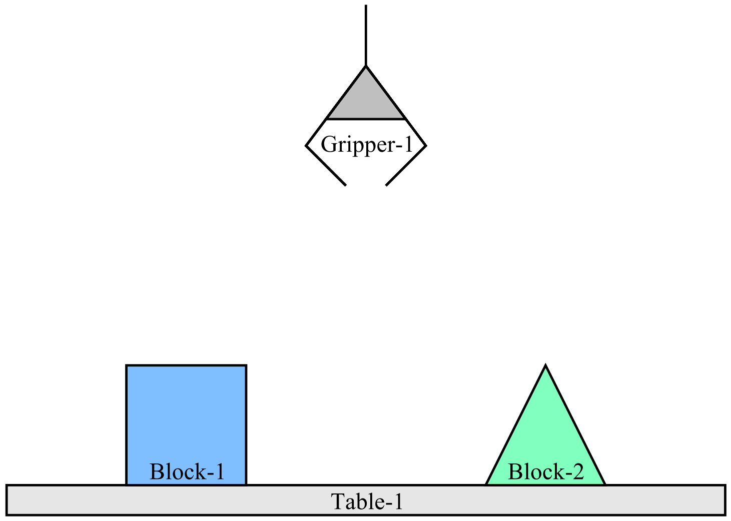


Figure 1: An example problem domain.

```
[defunk example-program []
  [move-right]
  [reach]
  [grab]
  [move-left]
  [drop]]
```

Figure 2: An example program.

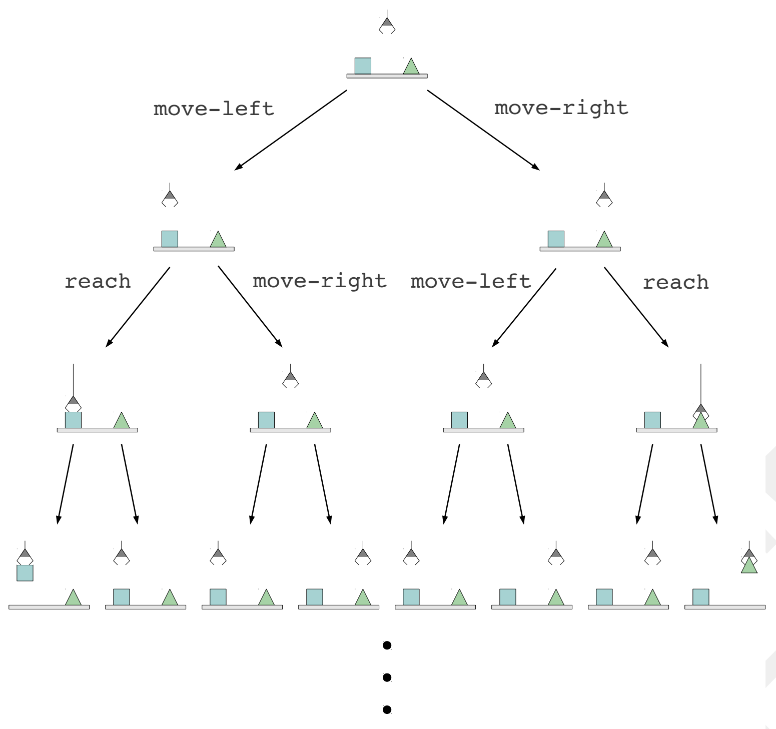


Figure 3: Exponential growth in planning as search.

1.6 STATE SPACE PLANNING

State space planning can be thought of as automatic programming. Planning refers to the computer's ability to write its own program. In an algorithm that plans by using a search algorithm, there is an initial state with a number of possible actions that may lead away from this state. We can move left; we can stop; we can move right. We can imagine what the possible future states would be after each action. In an imagined future state, the gripper may be in a different position. An example of the exponential growth resulting from continuing a planning search is shown in Figure 3. If every considered state in a search has the same number of possible actions, this search becomes an exponential search problem. The number of actions from each state is the *branch factor* of the search. Exponential searches quickly become intractable, even in state spaces with a reasonable number of actions. For example, games of chess are played to 20 levels deep with about 30 moves from each state. Thus, expanding this search tree gives 30^{20}, or 3×10^{29} plans. If there were one billion computers that each processed one billion plans per second in parallel, this search would still take over ten thousand years to complete, and that is just to play the first move! The fastest algorithms include good heuristics, e.g. a count of the number of pieces on the board for each player, etc.

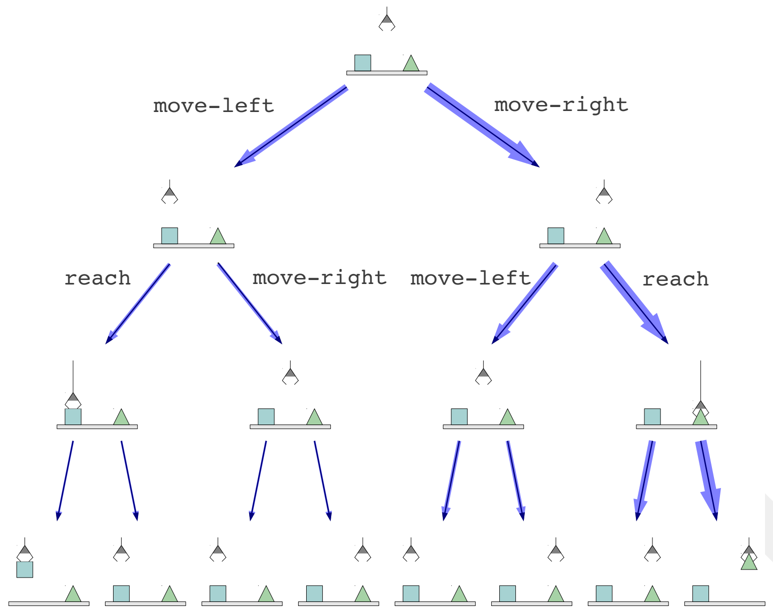


Figure 4: Exponential growth example with heuristics.

1.7 HEURISTICS

When a problem is exponential, it progresses for as long as can be afforded computationally, traversing the state space, hopefully finding the goal state in this large space of all possible alternatives, but then stops, usually not able to find the required answer to the problem in the given time or space limitations. *Heuristics* are one way to alleviate the problematic exponential growth of the search tree. In order to search more efficiently, heuristics provide weightings over the search tree that give a metric of distance to a completed plan. Beam search is an example of a heuristically weighted search that has a finite number of plans that it considers, forgetting the rest. As used in beam search, a heuristic is a function that defines an ordering on search paths. A heuristic function defines whether or not a given plan will successfully accomplish the goal, also, a heuristic gives a metric of the planning time required until a plan is created that will accomplish the goal. The algorithm uses this information to guide the search, thus reducing the branch factor of the search tree, reducing the search problem. Figure 4 shows the same exponential growth example with heuristic weightings overlayed with blue arrows.

1.8 REPRESENTING ACTIONS

The planning as search problem assumes that we have a representation of the world and a representation for the changes that actions perform. Given these models, the physical world can be simulated according to different plans. Fikes & Nilsson (1972) describe an action representation, called "STRIPS", that includes an object called a *transframe* for simulating actions. In the STRIPS model, the world is a set of symbols. A transframe is composed of two sets: one for the removals from the world and one for the additions to the world. A transframe represents a change between two states of the world. In a symbolic relational domain, the state space is a set of symbolic relationships rather than just symbols. Here is an example of a symbolic relationship that could exist in a relational domain: `[block-1 is-on block-2]`. Here is an example of a transframe for the world in a relational domain: *remove* `[block-1 on table-1]` and *add* `[block-1 on block-2]`. In predicting the effects of an action on the world, STRIPS considers one transframe for each action. Transframes can also be dependent on the current state of the world. Although many function approximation methods could be used, in my simulation model I have addressed the problem of learning to predict the correct transframes in the terms of Mitchell's (1997) "hypothesis spaces," which provide a simple and understandable formulation of the category hypothesis learning problem, given labelled examples. Hypothetical models are learned to predict the effects of actions. The physical state space informs sets of hypotheses that can be used to support assumptions, thus, the creation of new knowledge from an absence of knowledge, given the listed assumptions.

1.9 LEARNING TO PLAN FOR SUCCESSFUL EXECUTION

Dependency traces for a hypothesized successful plan creation compose an important set of knowledge to associate with a plan for debugging the planning process when, later, it is realized that the plan fails to execute. Figure 5 shows a picture of dependency traces with hypothesis creation events being pictured as shapes on a time line. These hypotheses have arrows between them that represent the derivation dependencies of each hypothesis creating decision. The circles in the picture represent the symbolic states of the world that are used to create hypotheses, which are represented by squares. If any hypothesis is used to derive another hypothesis, these dependencies give a credit assignment path that is able to jump back retrospectively an arbitrary distance in time as well as between reflective layers. Notice in the picture that the circle on the far left is a dependency of a square block, but these two events are not necessarily consecutive in time. If a hypothesis is derived from a number of other hypotheses, possibly far in the past, and this hypothesis fails in action, each one of these traced dependencies represents a learning opportunity. Every additional layer of reflective control in the model, represents another hypothetical learning opportunity, leading to more efficient search toward plans that succeed in execution.

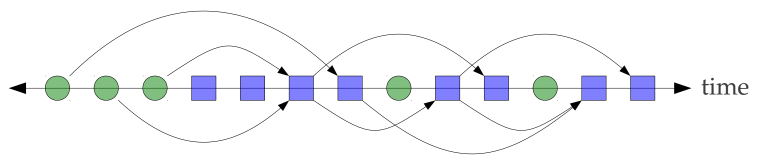


Figure 5: Dependency traces, where circles are symbols, squares are hypotheses, and arrows are dependencies.

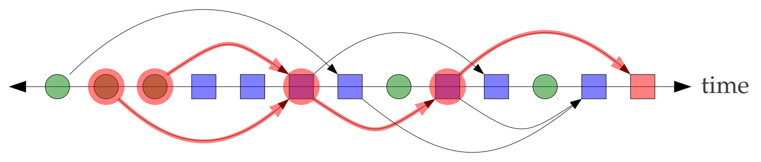


Figure 6: Tracing bug dependencies, where the bold arrows represent the credit assignment path for a failure in a hypothesis, represented by the red square on the far right.

1.10 FINDING A BUG

The credit assignment problem arises when a bug occurs in some part of the system. For example, plans can fail physically to accomplish what they have been previously hypothesized to accomplish. There are many knowledge dependency algorithms that work at this physical knowledge level. In this thesis, a layered reflective knowledge representation is presented for propagating failures to knowledge manipulation actions as well as physical actions. Figure 6 represents a bug found in a planning hypothesis. For example, the planning hypothesis could be that some certain planning activity leads from one type of plan to a plan that will successfully complete execution without failing. Failure propagates through reflective layers of goals and distinct classes of hypotheses. Bugs are propagated through every reflective layer and, therefore, each reflective layer is presented with a different learning opportunity.

1.11 COMPARING TEMPORAL AND REFLECTIVE LEARNING

A reflective learning algorithm implies arbitrarily better learning algorithms, including one-shot learning algorithms. In many domains, the opportunity to learn is rare. When the cost of failure is high, it is important to learn as much as possible from each failure.

I will refer to machine learning algorithms that learn by assigning credit for failures to the temporally previous action as *temporal learning* algorithms. Contemporary temporal learning algorithms are relatively advanced, some even having relational object-oriented models of actions and the world. Temporal learning algorithms learn by assigning credit for a failure to the previous time step or the previous action. A reinforcement learning

Learning Algorithm	Time	Space	One-shot Learning Opportunities
Temporal	$O(n)$	$O(n)$	n
Reflective	$O(n)$	$O(n * m^2)$	$n * m$

Figure 7: Temporal and reflective learning complexities with one-shot learning opportunities.

algorithm that learns in this immediately temporal sense is called the *temporal difference learning* algorithm as described by Kaelbling, Littman & Moore (1996). In temporal learning, the previous action is considered in the context of the previous physical state of the world. Preconditions and postconditions for the action are updated. In other words, the categories of the world are updated for this action, given the unexpected reward or punishment, a failure in expectations. In the temporal difference learning algorithm, there is a process of focusing on each previous action and state combination in temporal order to assign credit for failures. Temporal learning algorithms assign credit back, sequentially in time, from action to action.

Reflective learning algorithms can be thought of as having a temporal learning algorithm within each necessarily distinct layer of knowledge. Reflective learning can take advantage of concurrent processors for each separate layer of activity. Let m be the number of reflective layers in a learning algorithm with each layer on a concurrent processor. If there is a memory of actions and the state of the world for the previous n time steps, a temporal learning algorithm spends $O(n)$ time relearning the effects of the previous n actions. The reflective algorithm also spends $O(n)$ time but relearns the effects of $n * m$ actions, where m is limited by memory and processors. Also, if n is 1, this problem reduces to 1 learning opportunity for the temporal learning algorithm and m learning opportunities for the reflective learning algorithm. Figure 7 shows a comparison of the time and space complexities of temporal and reflective learning algorithms as well as the number of learning opportunities afforded by one failure.

1.12 REFLECTION

The term reflection is a commonly used word in computer science and AI. The idea is extremely simple and is the focus of this thesis, but, because of its simplicity, it is a widely applicable idea. In fact, Maes (1988) distinguishes over 30 different types of *computational reflection*, grounded in the computer science literature. The type of computational reflection that is introduced in this dissertation is not included in Maes' overview, although the implementation of this model is based on many of the forms of computational reflection that Maes does describe, e.g. procedural reflection, type reflection, frame reflection, and others. She does not mention the

The Model:	A foundation for how to think about reflective thinking.
The Simulation:	Introduces a discrete state space to the model for a mathematical description of reflective thinking.
The Implementation:	Introduces a computational transfer function to the model for automatically simulating the mathematical description of reflective thinking.

Figure 8: The derivative nature of the contributions.

type of reflection that I am focused on in this thesis because it was not, at the time, commonly considered computational. The type of reflection that is modelled here is a psychological type of reflection: the ability to think about thinking in addition to the ground problem. This psychological form of reflection is the focus of this thesis.

In terms of this model, reflection is the ability to perceive, act, and plan mental activities. A planning process is a type of mental activity, weighing decisions, making plans, and trying to accomplish goals. A process that optimizes this planning activity by, for example, learning heuristics for the planning search, is an example of a reflective process. In general, this layered reflective model learns in reaction to bugs and relearns the planning processes through dependency tracing for credit assignment. This model is similar to current models of *metacognition*, which will be described as related research.

1.13 THE DERIVATIVE NATURE OF THE CONTRIBUTIONS

The three primary contributions of the thesis are: (1) the implementation, (2) the simulation, and (3) the model of mind. My primary contribution is explained last because it is a derivative of the first two. The model of mind is the simplest contribution and serves as a foundation for the other two. The simulation model introduces a discrete state space to the model of mind, so that a mathematical description can be given of the model of mind. The implementation model introduces a computational definition of the transfer function for the state space introduced in the simulation model. Each of these contributions builds on the simpler, prior contribution. Therefore, in order to explain the implementation, I begin with the simplest contribution and work through each additional component in order to accomplish the primary goal of explaining the computational implementation of reflective thinking. Figure 8 shows the derivative nature of the contributions of the thesis in the order that they will be described from most foundational to most abstract.

Part I

THE MODEL

2 MODELLING THE CONTINUOUSLY DYNAMIC

You can't find love in a dictionary.

— Dennett (2009)

In this dissertation the focus will be a description of thinking that reflectively learns to accomplish goals. Part I begins with a clear distinction between dynamic continuity and static arrangements of symbols, which are *the* fundamental key distinctions in my model. The model of reflective thinking grows quickly within this initial foundational conception, and leads necessarily and naturally to the goals of this dissertation.

In Part II, the simulation, I will describe an example of a mathematical description of the model, which I will use to evaluate the model in terms of the performance of a search algorithm implemented in a reflective simulation.

In Part III, the implementation, an example of a computational implementation of reflective thinking will be described. The implementation is a layered cognitive architecture with three working layers. I will focus on a very simple physical block stacking problem domain as a tool for explaining simply, through examples, the layered classes of reflective learning. In describing the working model implementation, I will maintain an awareness of the explicit distinction between a static model and the dynamic referent.

2.1 SYMBOLS AND CONTINUITY

Symbols are static referential tools of thought. Words are examples of symbols. Mathematical expressions are composed of arrangements of symbols. All written language and spoken language is composed of sequences of symbols. While various fonts, writing styles, and speaking accents do influence the appearance of symbols, the fact remains that without symbols there is no language. A symbol is discrete and distinctly separate from other symbols. Symbols do not blend into one another when they are put side by side. Using symbols, models can be created. Therefore, the crux of my AI modelling problem is in understanding the continuous activity of reflective thinking. I have found understanding the reflective focus to be centered around understanding how to answer the following two questions:

1. What is continuous space?
2. What is the continuous activity that fills this space?

Any written or spoken answer to any question must be composed of discrete symbols. Some questions have clearly discrete and satisfying answers. For example, when asked, "What is $1 + 1$?", a satisfying answer is clearly "2". In this case, the question asks for another representation for the already symbolic focus of the question, "$1 + 1$". When we ask, "What is continuous space?", this is a question that refers to something that is not a discrete symbolic representation. Because any answer must be specified in discrete symbols, the answer can never be equivalent to the focus of the question. The focus of the question is not the words, "continuous space", but instead, the question is about the referent of these words. There is no exact discrete symbolic answer that is equivalent to the focus of the question. Answers, therefore, to a question about continuous activity are necessarily approximated with discrete symbolic language.

Answers to these questions abound, depending on the goals of the responder. My goal is to build an AI model that makes arbitrarily greater improvements given *any* non-reflective goal-oriented learning algorithm. In other words, my AI must be able to abstract and use models of its own continuously dynamic reasoning activities. The goal hinges on the AI's ability to deal with the inherent contradiction in answering these two basic and simple questions for itself. If it is understood that any answer must be based on language, then it becomes obvious that there is no general or final answer that can be predetermined for as yet unknown goals. Many philosophers, scientists, and spiritualists have provided different answers to these questions in order to accomplish different goals. Here are some caricatured answers to these two questions that apparently have very different explanatory goals:

- **Physicist:** Continuous space can be approximated by an infinite number of symbols that are arbitrarily close together. For example, if we have the integers 0 and 1, then we know that between these symbols, rationally, we can define the symbol $1/2$ that exists between them. We know that if we are describing continuous space that there must be a discrete point between any two given discrete points in our model. We use many dimensions of these symbolic numbers, in order to define laws describing the dynamic continuous potential and kinetic energies in multiple continuous spaces and times.

- **Spiritualist:** There is a continuous indivisible flux of energy that is the being of everything and nothing together as one. All beings are part of this energy that flows among and between us without any separations actually existing. We call this God, the energy that is in each and every one of us that contains all time and space, life and death, in one experiential existence.

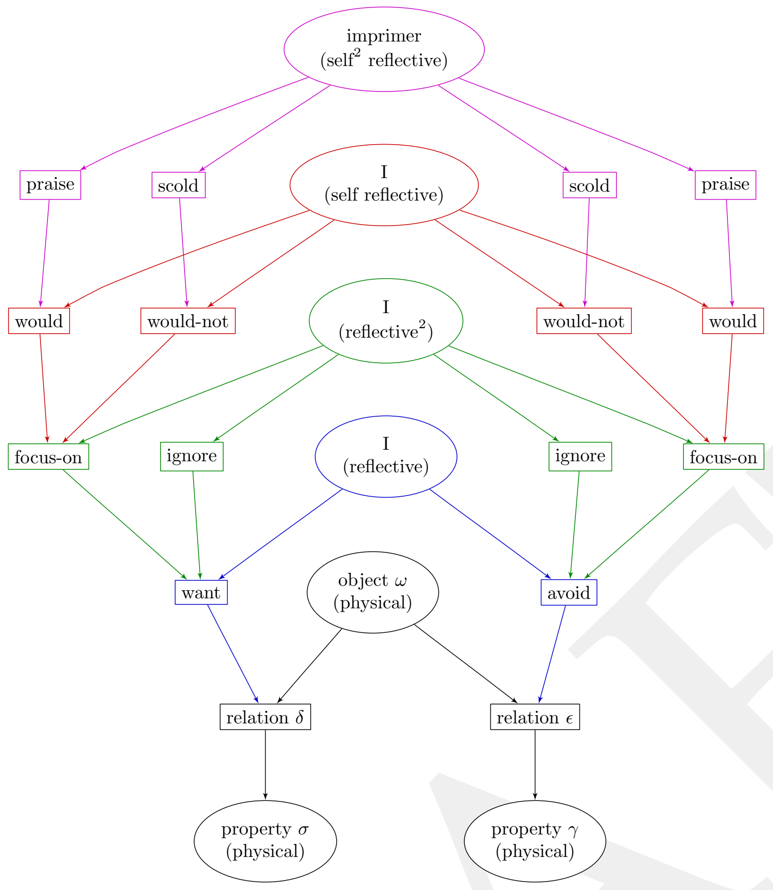


Figure 9: "The Treason of Images" ("This is not a pipe") by Magritte (1919) is an image of a pipe. The symbolic image is not equivalent to its actual referent, which is neither symbolic nor an image.

- **Philosopher:** The continuous amorphous perceptual experience is the real dynamic ongoing activity of being in the present, from which all discrete perceptions are abstracted. From these artificial, man-made symbolic abstractions, all models of discrete activities in the discrete dimensions of time and space are constructed.

Each of these caricatures, presents a very different model of the continuously dynamic activity in continuous space. The key idea to understand here is that each of these models *describes but is not equivalent to* this common continuous space and the dynamic activities that exist within it. Magritte (1919) visually explains this point with "The Treason of Images" reproduced in Figure 9. The different descriptions that are used are good for different goals, but they are all similar in that they are all based on symbols. They are all necessarily discrete descriptions of something that is not fundamentally discrete. The physicist uses an infinite number of symbols to describe this continuity. A model that includes an infinite number of symbols, like the mathematician's real number line, does not magically make this model not symbolic.

Models are all intentionally created, based on discrete symbols, in order to reference and describe this continuous space of continuously dynamic activities. A reflective AI, therefore, must be a model of the relationship between the goals of the responder and the creation of relevant models of continuously dynamic activity. In other words, a reflective AI, must explicitly be a programmed model of this fundamental modelling problem, a model of learning when a given model is good or bad for accomplishing its own goals. A reflective AI would be able to question the utility of these caricatured models of physicists, spiritualists, and philosophers, based on whether or not they help it to accomplish its own goals.

2.2 DYNAMICALLY GROUNDED DESCRIPTION

A grounded description references the dynamic. Symbols, words and language refer to undifferentiated amorphous perceptions or ongoing general experience and have grounding only in this way and create the tools with which the undifferentiated can be manipulated, understood, or communicated.

The further ability of symbols to also reference other symbols allows the additional possibility of creating a referential circularity in a static model. An illusion of a primitive grounding is possible in such a cycle because all of the symbols have a referent; however, because all references within such a closed cycle appear to be exclusively kept within the static model and do not seem any longer to refer to anything dynamic, there is a danger of there being no meaning other than circularity to such a purely static construct. Symbols are tools for modelling and are useful only if the circularity is not mistaken for a grounded reference. Thus, I strictly avoid presenting such illusions of grounding as primitive in this dissertation. To allow for grounded descriptions, I will first outline my conception of the dynamic, to which my description ultimately must refer. A description of reflective thinking referring to an *a priori* concept that has not been derived in terms of symbols allows for something to remain outside of the static model as the dynamic referent. This last step avoids the creation of a purely self-referential cycle and prevents illusions from becoming mistaken for grounded parts of the model. In the next section I will describe the necessary conception of the dynamic to which my description of reflective thinking will refer.

2.3 ACTIVITIES IN DURATION

Bergson (1910), in his seminal Time and Free Will, discusses the dynamic as heterogeneous activities in Duration. Activities in Duration will serve as my conception of the dynamic for my description of reflective thinking. Activities in Duration are the dynamic ongoing activity, heterogeneous activities that are neither distinct nor separate. There is a trick, a sort of slight of hand, required here in describing Bergson's conception of the activities in Duration because, even though referred to through symbolization, these activities exist independently of symbolization.

For example, when I look at a tree, I may say that the tree has green leaves and a brown trunk; the symbols, green and brown, refer to the activities in Duration, which I am seeing. When I focus carefully on the dynamic activity of my situation, I see that the leaves and trunk could be described by a greater combination of various greens, yellows, grays, browns and reds. Symbols limit perception to a discrete sort of projective presupposition. Therefore, as I introduce more symbolic references in my increasingly limited description of the dynamic, a growing danger of losing focus on something dynamic emerges, creating a cycle of symbols seemingly referring exclusively to symbols.

The motivation to tangentially define more clearly what I mean by the symbol green or brown by using more symbols results in a circularity from which there is no exit, and therefore, a false dilemma from which I will never again be able to see or talk about any dynamic tree as the referent. In order to avoid the danger of purely cyclic symbolic references in my model, I will continue to relate the parts of my model to the dynamic activity of reflective thinking, which exists as activities in Duration.

2.4 actual reflection

The activities in Duration are the given dynamic ongoing. These activities are the inseparable actions that are currently available to reflection. Reflection is the ability to symbolize this ongoing activity. For example, the activity of perceiving a table can be symbolized as the name "table". Similarly, the activity of walking, can be symbolized as present participle "walking". Note that my model of reflective thinking assumes that there is ongoing activity before any symbols exist for the actions of thinking to manipulate. Thinking is an auxiliary activity that manipulates static symbolic constructions that refer to the primary ongoing activity of existence. Thus, the activity of perceiving a table requires neither that this activity itself be reflectively symbolized nor that a secondary thinking activity manipulates the resulting symbols. I will use the phrase *actual reflection* to refer to the activity of symbolizing ongoing activities.

2.5 order of space

Activities in Duration may be symbolized in a homogeneous medium without activity. Bergson (1910) calls this medium Space, allowing symbolic references to activities in Duration to be related. This is the primary fundamental distinction in both the mathematical simulation and computer implementation of reflective thinking. Of major importance is the fact that one does not derive the other, but activity and order coexist with neither Duration being a derivative of Space nor Space containing the fundamental activities in Duration. Obviously, Bergson has gone through painstaking efforts to leave room in his description for the dynamic as some independent, non-contingent activity not derived from either Space or Duration. He makes the necessary concession that a description of something dynamic is only grounded in that it leaves room for this dynamic to exist independently of its own symbolization. This modelling work is meant to be similarly grounded in the necessarily separate dynamic activity of reflectively learning to accomplish goals.

2.6 MODELLING THE DYNAMIC

Duration and Space are the fundamental duality that forms the basis of my model of the dynamic activity of reflectively thinking. Thus this duality exists pre-reflectively, prior to symbols being created from the activities in Duration and prior to these symbolic references being related in Space. Note the fundamental contradiction of intentionally describing the dynamic only in static symbolic terms. For example, if Space has no activity and is not itself composed of activities in Duration, to what is the symbol "Space" referring? Symbol "Space" refers only to the logical potential for an absence of activity, the potential for itself as not existing; thus, it is the logical necessity of the negative implication of positive existence, to which no alternative can be assigned. Again, my model explicitly does not provide a definition of this implicit, fundamental and irresolvable contradiction. Thus, my description will appear to begin in the middle of something endlessly dynamic and ongoing, the *in medias res* of an underived time and space, maintaining an explicit awareness that static descriptions are only tools for dynamic reflections. Duration and Space are not placeholders for the dynamic; they are intermediate programmed fictions in the implementation that I use here to model the dynamic ongoing, the actual activity, which is not a static arrangement of symbols.

2.7 SPATIAL REFLECTION

The activities in Duration are the dynamic activity of being in the present. Symbols, like the photographic, are static references to this dynamic. Again, there is a fundamental distinction here between the dynamic and the static. Because AI already refers to an AI model, I use an asterisk notation to refer a model that the AI learns in simulation. In this way, Spatial arrangements of symbols can actually exist in the dynamic as an actively maintained static construction, a model* within the AI. The ongoing activities that hold symbols in a specific Spatial arrangement can be symbolically reified. I will refer to the activity of symbolizing the activities of a Spatial arrangement of symbols as *Spatial reflection* or *reification*. Spatial reflection is a form of actual reflection.

2.8 the physical layer

My model includes three layers of ongoing dynamic activity. I will use the phrase *physical layer* to refer to the pre-reflective layer of activities that exist necessarily prior to any subsequent reflective symbolization activity. The physical layer is the only pre-reflective layer of activity in my model. Therefore, a clear line is drawn between the physical layer and those activities that symbolize and Spatially arrange symbols. Thus, they are not included in what I will refer to as physical activity. Note that in most AI models, there is a distinction between the mind and the body or the environment. In my model, *all* activities that exist are referred to as *the mind*. [1] Thus, the dynamic mental activities that I am referring to in my model are not an object that is viewed subjectively. Critical to an understanding of the goals of objective science is that the creation of object and subject distinctions is allowed to be a part of the activities of the mind that can be simulated with my AI; because this understanding is so critical to an advancement of scientific understanding, I will clarify the mental derivation of objective science in Appendix A. Because the abstract creation of object and subject relationships is a relatively advanced mental activity and not key to my thesis, I will discuss this as future research in chapter 20, which would be a promising place to extend my model to include ideas like physical bodies and other subjective and objective environmental perspectives.

2.9 orders of reflective layers

In addition to the pre-reflective physical layer of activity, my model includes two additional layers of reflective thinking activity that use two different classes of causal models in order to accomplish two different classes of goals. I will use the phrase *first-order reflective layer* to refer to the reflective layer of thinking that creates symbols and Spatial arrangements that refer to physical activities. I will refer to reflectively symbolizing physical activities as *physical reflection*. Physical reflection is a form of actual reflection and is one of the primary reflective thinking activities; remember that physical activities are pre-reflective, so physical reflection is not itself a physical activity.

Obviously, the first-order reflective layer cannot create symbols or Spatial arrangements that refer to its own activities. The first-order reflective layer has the *a priori* ability to maintain its own actively reified Spatial relationships as existing between symbols that refer to physical activities. The lack of the ability

1 Because my model of mind is an absolute reference to the dynamic, it makes very few modelling assumptions. Because my model has been previously mistaken for the philosophy of "Subjective Idealism", I have included a description of the mistake of Subjective Idealism, and a clarification that my model does not make this mistake, an important difference between my model and this problematic philosophy; I explain in section 6.1.

for the first-order reflective layer to refer to its own symbolically creative or Spatially manipulative activities acts as a sort of protective "firewall" that clarifies my model by keeping clear distinctions between each class of causal knowledge in each subsequent reflective layer of thinking. Before describing distinctions between classes of causal models, I will spend the next few sections describing the composition of temporal transitions and causal hypotheses.

Reflection over the activity of the first-order reflective layer necessarily creates a second layer of reflection and thinking. Thus, clearly deriving and describing this second layer of reflective thinking additionally necessitates an implicit and arbitrary number of layers. This condition is the focus of this dissertation. Minsky (2006) describes a six-layer reflective model of thinking about thinking. The first three layers of my model and implementation, including these two layers of reflective thinking, can be compared to the first four layers of Minsky's model. There are many close parallels between my model and Minsky's model, and I will describe these modelling parallels in chapter 7.

2.10 CONSCIOUSNESS, AWARENESS, AND EXPERIENCE

There are many terms that are used in the literature when referring to the concept of the dynamic. Each of these terms has a different associated collection of necessary contingencies that, as isolated and independent, tend to become absurdities. For example, the general term consciousness always begs the question *of what* one is specifically conscious. There is an implicit independence of subject and object when using this transitive term, which is not assumed when the term is used as a reference to the dynamic.

The terms awareness and experience are used similarly and have the same logical contradictions as the term consciousness when they are considered as independent isolated references to the dynamic. Each of these terms has different meaningful uses when they are not used in this independent sense. For example, in this dissertation an awareness of the potential confusions these terms introduce when they are used singularly in abstract isolation as reference to the dynamic will be maintained.

Thus, I will not argue that these conceptions of the dynamic are incorrect. Rather, they are limited to particular and less generalized models that are restricted descriptions of the dynamic. In this dissertation, however, in order to avoid confusion, I will never use these terms in their limited and abstracted sense. Instead, I will conflate all of these uses of these terms as all being the same as my conception of the dynamic, the ongoing activity that my model accepts, symbolizes, and thinks about as given.

2.11 EXTENSIVE AND INTENSIVE QUANTITIES

Sometimes it is hard to imagine how a fundamentally symbolic model can ever refer to the "intensities" of perceptual stimuli. Thus, in this section, I make a distinction between two types of intensities, the "extensive" and the "intensive". This distinction is used to explain how the basic symbolic component of the model can come to represent various aspects of everyday thinking that, according to common sense, are fundamentally of a certain intensity.

Bergson (1910) makes a key distinction between comparisons of "extensive" quantities and comparisons of "intensive" quantities. Extensive quantities are Spatial comparisons. *Extensive quantities* are between two symbolic references that share a containment relationship in terms of their referents: one symbolic referent is inclusively contained by another symbolic referent in Space. For example, one body may be said to be larger than another in terms of its Spatial extent. Intensive quantities are more subtle and an example is in terms of pain, which can be said to be more or less intense. *Intensive quantities* are similar to neither numerically nor Spatially extensive containment relationships but, instead, refer to different fundamental activities in Duration. For example, a pain that has no intensity is not a pain at all; a pain that has a mild intensity may be equivalently referred to as an irritation or an itch; a pain of high intensity may be equivalently referred to as a sharp pain or a throbbing pain. The point is that these intensities of quality are actually very different from extensive quantities in that they are not greater than or less than one another in the sense of a containment relationship. For example, a very intense pain, such as a shooting pain or a throbbing pain, does not in any sense contain a lesser mild pain, such as an irritation or an itch. This same pattern exists with detailed descriptions of other qualities that are often thought of as having comparable intensities: heat and cold; light; pressure; sound; pitch; aesthetic feelings, such as grace and beauty in music, poetry and art; emotions, such as rage and fear; moral feelings, such as pity; affective sensations, including pleasure, pain and disgust. Each of these examples points out that the apparent relationships of greater than or less than with respect to intensity are static Spatial arrangements of different fundamental activities in Duration.

2.12 NUMBER

Some models make the complicating assumption that numbers are an additional component to every symbolic perception just for this purpose of modelling intensity. In this section, I describe how numbers are related to the model: numbers are thought of in my model as arrangements of symbols in Space.

Bergson (1910) shows the derivation of all numerical representations as requiring an accompanying extension in Space. For example, in referring to the individual sheep in a flock of sheep, he explains counting:

> Number may be defined in general as a collection of units, or, speaking more exactly, as the synthesis of the one and the many...
>
> No doubt we can count the sheep in a flock and say that there are fifty, although they are all different from one another and are easily recognized by the shepherd: but the reason is that we agree in that case to neglect their individual differences and to take into account only what they have in common. On the other hand, as soon as we fix our attention on the particular features of objects or individuals, we can of course make an enumeration of them, but not a total...
>
> Hence we may conclude that the idea of number implies the simple intuition of a multiplicity of parts or units, which are absolutely alike.
>
> And yet they must be somehow distinct from one another, since otherwise they would merge into a single unit. Let us assume that all the sheep in the flock are identical; they differ at least by the position which they occupy in space, otherwise they would not form a flock.

The last point here illustrates the idea that numbers are always derived from symbols situated in Space. Therefore, a model for thinking based directly on symbols cannot, by definition, be more complicated than a model based on a numerical representation. A number is, after all, only the reified result of an axiomatic construction of Spatial organization of symbolic references to the dynamic. In other words, they are specific sheep of the dynamic. The explicit implication is, therefore, that, in building an AI that is capable of reflective thought, the model building process and its use of numerical representations must be explicitly available for inspection by the AI itself. Here, I employ Occam's razor to simplify the loop of reflective representation in order to ease building the first proof-of-concept examples of AIs capable of reflective thinking.

2.13 TIME AS SPACE

Symbolic references can be ordered Spatially. These symbolic relationships can be reified and treated symbolically by further ordering Spatial relationships. Thus, Bergson explains time as a form of Space:

> Now, if space is to be defined as the homogeneous, it seems that inversely every homogeneous and unbounded medium will be space. For, homogeneity

> here consisting in the absence of every activity, it is hard to see how two forms of the homogeneous could be distinguished from one another. Nevertheless it is generally agreed to regard time as an unbounded medium, different from space but homogeneous like the latter: the homogeneous is thus supposed to take two forms, according as its contents co-exist or follow one another. It is true that, when we make time a homogeneous medium in which conscious states unfold themselves, we take it to be given all at once, which amounts to saying that we abstract it from duration. This simple consideration ought to warn us that we are thus unwittingly falling back upon space, and really giving up time.

Time exists as a static creation, a derivative arrangement of symbols in Space. Understanding time to be static is important because this allows the dynamic heterogeneous activities of Duration and a static homogeneous unqualified Space to remain, respectfully, neither distinct nor separate. By prohibiting this separation, and, thereby, their independent existences, I avoid defining the dynamic activities in Duration in terms of static symbols, which, in turn, avoids the creation of a self-referential cycle, thus sidestepping the potential illusion of a false dynamism. Being clear in our understanding of time used intentionally as a static tool of thought avoids confusing the focus of the modelling process to focus on modelling the dynamic. Again, a model that is purely of a model leads to an AI based on a tautology because models are only grounded in that they reference the dynamic.

2.14 temporal reflection

Spatial arrangements of symbols can be used to represent a past through memory and subsequently project that distinction as an analogous "future", through imaginative action, thus constructing a derivative conception of time. The past and the future, therefore, remain static constructions that are actively maintained by the model as Spatial relationships in the dynamic present. I will refer to the reification of Spatial relationships that order the past and the future as *temporal reflection*. Sequential time is created through temporal reflection. Thus, temporal reflection is a form of Spatial reflection. Like Spatial reflection, temporal reflection is also a form of actual reflection, symbolizing the active maintenance of a temporal Spatial relationship between static symbolic references.

2.15 simultaneity and transition

Correlation requires two symbols to be related in Space. As symbols are correlated in the Space called time, these temporal correlations are referred to as either simultaneities or transitions.

Simultaneities represent two symbols that are both actively symbolized in Duration. Transitions represent the change from the past to the future. Transitions can be extended in time in order to create counterfactual temporal extensions, called inferences. Inferences of the past and the future can be created by matching and repeating these transitions. A transition implies the removal of one symbol and the addition of another as a progression is made through a temporal sequence. If correlations are counted and compared in ratios, then these are called probabilistic correlations, and the resulting inference would be a probabilistic inference.

3 LEARNING CAUSAL HYPOTHESES

3.1 RESOURCES

Activities in Duration can be referred to symbolically by the reflective thinking layers. While all activities in all layers are activities in Duration, these activities can be symbolized as potentially actionable parts of plans by the reflective thinking layers. I will refer to a symbolic reference to activities that can be put into plans as a *resource*. Resources exist in the thinking layers as symbolic references to activities in the layers below.

3.2 ACTIVATION AND SUPPRESSION

In my model, I've included a logical idea that I refer to as *suppression*. Suppression is a symbolic relationship to a resource that can be put into plans. The idea of suppression is a subtle point with respect to the activities in Duration. The basic problem is that the activities in Duration are the activities that exist independently without thinking existing as any implicit, co-existent necessity. Further, the activities in Duration cannot be inactive, by definition. This is important. In other words, a symbolic reference to something inactive would imply a symbol that refers to something that does not exist, or, in other words, only exist as a contradiction. Therefore, the logical idea of suppression is only a static tool of thought, part of the thinking layers, entirely separate from physical activities. The subtle point here is that suppression does not disable the potential for activities. Suppression is only a logical Spatial relationship including the symbolic reference to the physical activity in question. I will refer to the logical alternative to suppression as *activation*. Activation and suppression refer to types of assumed Spatial relationships that are maintained between symbolic resources. Therefore, it would not be correct to say that physical activities have been activated or suppressed, but alternatively, it would be correct to say that a resource is in an ongoing, unstoppable, dynamic Spatial relationship that has activated or suppressed qualities. The activation and suppression of resources occurs in the sense of actively creating Spatial relationships that represent the logical qualities of activation or suppression, which have the potential to be logically consistent or contradictory.

Therefore, in my model, a resource can be both either activated or suppressed, which does not, by itself, imply a resultant activity in the layer below; logically, if a resource is activated and is not otherwise suppressed, then the resource is considered logically to be activated; however, when a resource is both activated and suppressed simultaneously, this is a logical failure that is cause for a plan to halt execution. Before discussing potential responses to plans failing in this way, it will obviously be necessary to first discuss the basics of the planning process.

3.3 CAUSE AND EFFECT

Two symbols correlated in time are not enough to compose a causal relationship. Two symbols correlated in time are simply a coincident transition from the past to the future. A causal relationship supposes an additional component, a necessary connective symbolic reference to the activities that are ongoing during the transition. Thus, the effect of the causal component becomes the transition itself. A causal relationship, therefore, has three parts: the symbolized activities in the present, the symbolized activities in the past, and the symbolized activities in the future. I will sometimes refer to these three parts of the causal relationship more succinctly as (1) the cause, (2) the necessity, and (3) the result. When causal models are used for planning, the symbolic reference that is the cause is referred to as a resource.

3.4 GOAL ACTIVITIES

It is important to understand that, in my model, goals are not derivatives of symbolized perceptions. Symbolized goals refer to the fundamental activities in Duration that give *a priori* direction to the activities of thinking. Goals can thus be arranged in Spatial orders, according to preferential qualities; i.e. goals can be considered to be positive or negative, something to seek or something to avoid.

In my model, a goal is a reference to activities, similar to a perception, except that it is arranged preferentially among other goals to be sought or avoided.

The first-order reflective layer symbolizes goals from the physical layer activities. This creation of a symbolic reference to goal activities becomes the reason for planning and acting toward or away from the associated perceptions. Note that because symbolic goals, perceptions, and resources in my model are static and do not derive from one another, a process of refining symbolic references to perceptions and resources can be undertaken in the pursuit of creating more accurate causal models that predict the activities in Duration that symbolic goals reference.

3.5 CAUSAL HYPOTHESIS

When goal activities are symbolized, causal hypotheses can be created for predicting the goal activities in the future. For example, if there are ongoing activities happening simultaneously with the symbolization of the goal, these can be hypothesized as causes of a future symbolized goal. Remember that a causal model has three parts: present cause, past necessity, and future result. In this case, the goal is placed in a future context, the symbolized current activity is in the present, and there must also be a past symbol to fill the last slot in the model.

3.6 limitations of logical goals

Logical approaches to AI have seen the value in considering goals to be relative arrangements of perceptual symbols. These sorts of symbolic relationships in my model are thought of as occurring simultaneously with, before or after goal activities. Static constructions that are actively maintained in Spatial arrangement can be reified as symbolic perceptions, but fundamentally the goal does not refer to a static construction, despite the possible correlation of goals with such constructions. In this way, my model gains the ability to think about and refine the symbolization of perceptions because symbolic goals refer to the activities in Duration, instead of the inversely limiting assumption that goals are specified in terms of static constructions of perceptual symbols. Static constructions are useful tools for accomplishing goal activities; however, reflectively, all static constructions, including symbols, are caused by the AI itself and the correspondingly responsible activities can change. Thus, all symbolic constructions are explicitly available as static references that are available for inspection by the AI itself. In other words, in purely logical approaches to AI, symbols are not understood to be static creations of the AI itself and, thus, cannot be reflectively debugged when they are wrong.

In order to allow these logical approaches to adapt, I see no alternative but to acknowledge the dynamic, allowing the potential for debugging grounded symbolic constructions. AI systems must use symbols in full awareness of their creation as changeable static references to a dynamic unstated actual existence.

3.7 physical goals

The first class of goal that I will discuss is called the *physical goal*. Physical goals refer to dynamic activities in Duration that are a cause for the creation and refinement of static symbolic perceptions and resources in the reflective thinking layers. These symbolic perceptions and resources are thus physical as well.

3.8 reflective classes of causal models

The first-order reflective layer creates causal models from physical perceptions, resources, and goals. I've described previously how goals can cause reflective thinking to create causal hypotheses. Another way to state this more succinctly is as follows: *goals are the cause of causal hypotheses*. Note two different meanings of cause in the previous sentence. In the latter case, I'm referring to causal models relating physical perceptions and resources, and in the former case, I'm referring to the first-order reflective activity that causes the creation of the physical model.

Allowing activities of the first-order reflection to be available for inspection by the AI itself allows my model to represent the transitions caused by first-order reflective activities. These transitions are knowledge level changes. By using this reflective technique, a new class of causal model is created, categorically different from the physical causal models that the first-order reflective layer manipulates. When the activities of the first-order reflective layer are reflectively symbolized, my model can then be motivated to learn to accomplish or avoid knowledge level goals, using knowledge level causal hypotheses.

3.9 PROBABILISTIC CAUSAL MODELS

Probabilistic models are an advanced and important thinking tool. Building a probabilistic causal model involves counting symbolic perceptions, resources, and goals. For example, let us consider that two causal hypotheses have been created that reference the same symbolic cause and the same symbolic perception in the past; let us say that the only difference between these two hypotheses is that they refer to two different symbolic goals in the future. This situation gives my model a reason to further distinguish symbolic representations for perceptions and resources, introducing more refined symbols for these perceptions and resources in order to lead to causal models that are more useful for correctly predicting these two different symbolic goals. This would be the more appropriate course of action if we wanted to correctly predict the symbolic goals; however, there is an opportunity here to build a probabilistic model that does not refine the symbolization of perceptions or resources. For example, both of the two causal hypotheses could be considered together to compose a probabilistic causal hypothesis. Using this probabilistic hypothesis, a future inference could be created that includes both symbolic goals, each with one half of a potential existence. Probabilistic causal hypotheses are useful for predicting the average number of times that a symbolic event will occur. Note that probabilistic causal hypotheses require counting and creating ratios from the more fundamental non-probabilistic causal hypotheses from which they are constructed.

4 LAYERS OF REFLECTIVE THINKING

4.1 FIRST-ORDER REFLECTIVE THINKING

After constructing causal hypotheses that predict the necessities and results of activating resources, these causal models are used to construct larger structures called plans. Plans are combinations of causal models that contain inferences of past necessities and future results. For example, one could imagine a sequence of resources that leads through a counterfactual future sequence of perceptions, resources, and goals. In the literature, these counterfactual constructions are referred to by a number of names, including: case, explanation, fiction, narrative, and story. Stories and narratives often include a social self-reflective representation, which I see as future work in the development of my model, which I will discuss in chapter 20. I will simply use the term *plan* to refer to all of these constructions that combine causal models into counterfactual views of the past and future.

First-order reflective thinking is the ongoing activity in Duration that is limited to manipulating symbolic references to physical activities. First-order reflective thinking symbolizes physical activities, creates physical causal hypotheses, and uses these hypotheses to create plans toward or away from physical goals. Further, first-order reflective thinking activities execute these plans composed of physical resource causal symbolic references.

4.2 n^{TH}-ORDER REFLECTIVE THINKING

I've previously described three layers of activity in my model: the physical layer, the first-order reflective thinking layer, and the second-order reflective thinking layer. For brevity, and to emphasize the layered nature of my model, I will sometimes use a superscript notation "reflective^n" to refer to the n^{th}-order of a layer in my model. For example, the reflective^1 layer will refer to first-order reflective activities, and the reflective^2 layer will refer to second-order reflective activities. Although the physical layer is not a reflective thinking layer at all, its place as the prior reference for the first-order reflective thinking activities moves me to inductively extend this notation to allow referring to the physical layer as the reflective^0 layer or the *zeroth-order reflective layer*. Zero thus serves as the implicit origin of an infinite regress of n layers. In this sense, the implementation of my model of reflective thinking includes activities of reflective order zero through reflective order two.

4.3 LIMITATIONS OF FIRST-ORDER REFLECTIVE THINKING

The act of creating a symbolic reference to physical activity is not accessible to the first-order reflective layer. This is critical to an understanding of my model. A creative first-order reflective layer action, such as creating a plan toward a physical goal, is limited to creating and Spatially arranging symbols that refer to physical activities. The first-order reflective layer cannot manipulate symbols that refer to its own activities without reducing itself to something hopelessly contradictory.

Another critical point to understand is that first-order reflective thinking does not directly manipulate physical activities. Neither symbols nor Space are physically actual; instead, symbolic references to physical activity and the relationships that order them in Space are effects and not causes. As static creations of an independent first-order reflective layer they are only derivative. First-order thinking is about physical activities and is limited to the terms of its own created static symbolic references to the actual physical layer. In contrast, the physical layer does not include any of the static constructions of thinking, neither symbols nor Spatial arrangements. This is because the physical layer does not include thinking, which is fundamentally a reflective activity.

4.4 SECOND-ORDER REFLECTION ON SYMBOLIZATION

First-order reflective thinking activities create, Spatially arrange, and otherwise manipulate symbolic references to physical activities. The activity of symbolization is a primary aspect of every reflective thinking layer; however, given the logical limitation that reflective thinking layers do not refer to their own activities with the exception of active Spatial arrangements, a second-order reflective thinking layer is necessarily required for creating symbolic references to the activity of symbolization. Note that the activity of reflecting on symbolization appears in all layers above the first-order reflective thinking layer, but this activity of reflecting on the activity of symbolization necessarily first appears only in the second-order reflective thinking layer.

4.5 NON-EXISTENCE OF SYMBOLIC REFERENCES

Reflecting on symbolization is one of the primary activities of the second-order reflective layer, resulting in the primary second-order symbols that refer only retrospectively to first-order activity. Having a symbolic reference to first-order symbolization allows the consideration of this activity to be symbolized as a second-order resource. Considering first-order symbolization as a second-order resource thus allows a second-order causal hypothesis to be created with this resource as the ongoing cause of a transition. This is key. In other words, the transition is from the “non-existence” to the purely retrospective necessity of the first-order symbolic reference to physical activity.

Note that non-existence only fulfills the logical function of a placeholder in the causal model, which refers to the necessity of an alternative to creation and existence itself. Thus, I’ve begun my description with the assumption that what exists is my reference, and that what exists does not require symbols to refer to itself or describe itself. However, in reflecting on and symbolizing the activity of symbolization, I have implicitly assumed this placeholder, “non-existence”, in the causal model that allows second-order reflective thinking about the existence of first-order symbolic references. Non-existence refers to the past slot in the second-order causal model of first-order symbolization. Causal hypotheses and their symbolic parts are static constructions, so non-existence can only be a reference to part of a static construction. Non-existence thus becomes a necessary modelling tool for second-order reflective thinking. Non-existence does not refer to a primary activity in Duration, non-existence refers to a static Spatial arrangement of symbols in the second-order reflective layer.

4.6 planning symbolic refinement

There is a very useful opportunity here to temporally extend the causal model in the future in order to predict the creation of new symbols based on previous symbolized perceptions or resources. This could lead to second-order reflective thinking creating plans for the first-order thinking layer to create new and more refined symbols that may be useful for either accomplishing or avoiding physical goals. This is a useful and powerful type of second-order reflective planning.

4.7 the distractive nature of non-existence

Non-existence as a tool of thought is as dangerous as it is powerful. The danger is that non-existence can be incorrectly considered to be *the preexisting dynamic*. In my model, I have defined the dynamic to be everything that exists as ongoing in Duration, maintaining an awareness of the absurdity of symbolizing something that is prior to symbols. Said another way, the danger is in forgetting that non-existence is a reference to the past slot in a static causal construction of at least order two. This danger of mistaking non-existence for the prior dynamic is insidious, distracting, and tempting because of its false promise of an explanation for creation and existence itself.

For example, the first-order reflective layer can create the perceptual symbol “green” to refer the current undescribed physical activities in Duration. These ongoing physical activities do not contain symbols and I have been very careful to not attempt to provide a description of that which cannot be defined in terms of static symbols, the activities ongoing in Duration, which are neither static nor in static terms. Therefore, the symbol “green” is a static creation that refers to these undefined activities, which are the given activities in Duration. They require neither symbolization nor thinking to exist.

By means of this new methodological notion of non-existence, a second-order reflective thinking activity is now able to create a causal model of this first-order symbolization, the creation of the symbol “green”. This would not have been possible if non-existence were allowed to function as a form of “pre-existing dynamic”. In other words, this second-order activity implies that before the creation of the symbol “green” there was a paradoxical existence of the non-existence of the symbol “green”. The fact is that the non-existence of the symbol is actually only the secondary result of the creation of the symbol, so we must not be confused when the second-order causal model places the non-existence of the symbol in the past slot. In this way the model remains cognizant of the fact that non-existence of the symbol is only a static tool of thought. “Active non-existence” is equally as absurd as “inactive existence”; actual non-existence is a tempting but simple contradiction.

4.8 third-order reflection on existence

The second-order reflective layer creates causal models of the creation of symbolic references. These second-order static constructions contain the primary references to existence and non-existence. Although my implementation does not include a third-order reflective layer, my model includes the potential for an arbitrary number of layers. In order to briefly describe the potential for third-order reflective thinking, let us consider a causal model that would be the creation of the third-order reflective layer.

For example, let the second-order activity of creating a causal hypothesis be symbolized as a symbolic resource in the third-order reflective layer. This symbolic resource could refer to the second-order creation of a causal hypothesis describing the first-order creation of a symbolic reference to physical activity. The symbolic resource that refers to the first-order creation of the symbol is placed in the past slot of this third-order causal model. In the future slot of this third-order causal model is a symbolic resource that refers to the second-order creation of the causal model that introduces the concept of existence being created out of non-existence. Note that the power of third-order reflection allows the creation of the causal hypothesis that correctly places the creation of the symbol prior to the simultaneous creation of the knowledge of the existence and non-existence of the symbol.

4.9 "sub-symbolic thinking"

I have previously discussed the danger of having the second-order reflective ability to create a causal model that predicts the existence of symbolic perceptions from non-existence. The danger is in believing that the non-existence of a symbolic perception represents some kind of actual pre-reflective mechanism that produces symbols. It doesn't. There is no such thing as a pre-reflective or pre-symbolic symbol or model. Reflection creates symbols and models that refer to the dynamic. In other words, models must necessarily model something other than their own activity. Thus, it is important to understand that a causal model that predicts the creation of symbols is the result of a second-order reflective activity, more abstract and static than the first-order creation of the symbol.

Many causal models that have been developed to describe the creation of perceptual symbols. Many of these are based on the static extension of physically objective assumptions, therefore allowing one to dissect brains and build models of perceptions based on such things as neural networks. These neural network models are sometimes said to be "sub-symbolic". It is important to realize that the integration is totally illusive, that the idea of a model being sub-symbolic is a contradiction in terms because all models are built from symbols. These so called sub-symbolic models are actually, at best, only at the level of second-order reflection, which is the first level of reflection that is able to model the creation of a symbolic perception. I think that the current trend in referring to these types of models, that in fact only actually appear to the second-order reflective thinking layer, and that further claim to be able to predict perceptions as sub-symbolic, is a confusing and distracting phrase in the field of AI. How to think reflectively about neural networks has not yet been described in the field of AI because neural networks are very complex numerical models that require at least a second-order level of reflective thinking, if they are to be understood at all as a model of the creation of symbols.

5 LEARNING FROM SUCCESS AND FAILURE

5.1 PLAN EXECUTION

First-order reflective thinking creates causal hypotheses that describe how physical resources may cause transitions between physical perceptions and goals. Additional first-order reflective thinking activities combine these causal hypotheses into plans that are supposed to accomplish or avoid goals, given their preferential arrangement. After a plan is created, its sequential instructions for activating or suppressing the resources contained in the causal slots of the hypotheses can be traversed and the causal resource that has been referenced can be put into the appropriate activated or suppressed Spatial arrangement. I will refer to this sequential process of traversing through a plan and following its activation or suppression instructions as *execution*.

5.2 PLAN INSTRUCTIONS

I've previously discussed the necessity of using activate and suppress instructions that relate resources. One instruction that has only been implicitly introduced at this point in my description is the model's ability to Spatially arrange a sequence of instructions, which I will refer to as the *program* instruction. There is also an instruction that allows the Spatial arrangement of concurrent instructions in the model, which I will refer to as the *parallel program* instruction. The program and parallel program instructions allow for the arrangement of hierarchies of planned sequential and parallel program executions.

I will give a more detailed description of plan instructions in Part III, but for now I will focus on describing how failures can lead not only to learning better first-order causal hypotheses for predicting physical goals from physical resources and perceptions but also to learning better second-order causal hypotheses for predicting first-order goals, such as the creation of a plan that will not fail when it is executed.

5.3 FAILURES AND SUCCESS

There are many types of failures that can occur during the execution of plans. Each different type of failure that occurs may have types of activities that, reflectively, will appear as the causal result of the failure; actively halting the plan execution allows for a debugging response activity to occur at the layer above the failed plan. I will refer to the execution of a plan that results in no failures as a *successful execution*. Plans may be considered to exhibit various types of success only in the sense that they have avoided the various types of execution failures.

5.4 activation failure

If we imagine that two plans are executing concurrently, I will refer to the simultaneous activation and suppression of a resource as an *activation failure*. An activation failure can also occur if two plans attempt to simultaneously activate the same resource. When an *activation failure* occurs, one possible response is to create a new plan to accomplish the combined goals of the two plans, but which includes the two plans as executing in sequential order.

5.5 compiling plans to resources

I've discussed previously how the instructions in plans can be sequentially interpreted as one form of execution. When a plan is found to fail when executed concurrently with other plans, it is sometimes helpful to consider the execution of a specific plan to be a resource in itself. I will refer to the process of turning the execution of a specific plan into a new resource as *compiling*. With the ability to compile plans to resources, we have another possible solution to activation failures. For example, if one of the conflicting plans is shorter or less critical than the other, the shorter plan can be compiled into a resource that is inhibited whenever the longer plan is executing.

5.6 expectation failure

As a plan executes, the results of resource activations can be found to not match currently symbolized perceptions. This failure of a causal hypothesis to correctly predict the transitional results of a resource activation I will refer to as an *expectation failure*. The most obvious response to an expectation failure is to correct the causal hypotheses in the first-order reflective layer that predict physical perceptions. The correction of physical causal hypotheses involves both eliminating the causal hypotheses that made the incorrect prediction and creating new causal hypotheses that correctly predict in light of the new transition between physical perceptions.

5.7 learning below and above the failure

Correcting the physical causal hypotheses that lead to expectation failures will ensure that the same type of failure will never be planned again. This is useful. Nevertheless, given reflective layers, the implementation can still be improved in at least two ways. Assuming that the causal hypotheses in the first-order reflective layer have failed, we can also consider corrections that can be made both "below" and "above" the failure.

For example, the causal hypotheses in the first-order reflective layer are based on symbols that refer to activities in the physical layer. Since, as already discussed, there necessarily are no symbols in the physical layer, the only allowed descriptions of physical activities must be created by first-order symbolization activities. Thus, one way to create more refined causal hypotheses that describe physical activities is to create more refined symbols that refer these physical activities. This parenthetical ability to refine symbolic references of an otherwise undefined activity is a critical feature of a reflective model of mind. Despite the fact that the refinement of symbolic references occurs in the same layer as the failed causal hypothesis, I will refer to this form of learning as being *below* the failure because it changes references to the layer below the failure.

Alternatively, we can focus on learning in the layers above the failure. The second-order reflective layer can think about the planning activities in the first-order reflective layer that created the plan that failed. Causal models of planning activities that lead to successful or failed plans can be used by the second-order reflective thinking layer to better plan sequences of planning activities. The second-order reflective layer considers first-order activities as resources that create plans that, when executed, result in either successfully or unsuccessfully accomplishing physical goals. I will refer to correcting causal hypotheses in the layers above the failure as learning *above* the failure.

Learning above the failure is logistically difficult because of the arbitrary amount of time that may have passed between the creative activities that result in plans and the sometimes much later execution of the plan that results in failure. This thesis and its implementation are exclusively focused on this type of learning. In other words, the failure becomes above the failure in response to an expectation failure.

5.8 first-order reflective thinking activities

I've previously described physical goals that are symbolic references to physical activities that can be preferentially ordered. The first-order reflective layer activities create causal hypotheses and put these together into plans. This leads me to a critical point: I have been purposefully vague in describing the types of plans that are created by the first-order reflective layer. In my model of mind, I have statically divided the ongoing activities in Duration into layers. Despite my division, which allows talking about orders of reflective thinking that build layers of abstractions upon other abstractions, in the dynamic, all of these symbols or Spatial arrangements only exist actively, not specifically prioritized statically as discrete and separate components of action. Thus, all layers of activity in my model are given the appearance of being *in media res* as ongoing, inseparable, and simultaneously active activities in Duration. So, while physical goals may be symbolized by the first-order thinking activities, the first-order thinking activities are also making plans. Thus, the types of plans that this activity creates are undefined by priority in this sense. Reflectively, from the perspective of second-order reflective thinking, we may see clear causal hypotheses that give a reasonable understanding of the first-order reflective layer activities, but these activities are seen as "causal" only in retrospect and not in execution. Thus the idea of causal can be understood as introducing a separate and reflective point of view to the idea of execution. Thus, the results of planning activities in the first-order reflective thinking layer can be reflectively learned by the second-order reflective thinking layer.

5.9 first-order reflective thinking goals

The second-order reflective layer learns causal hypotheses of the given first-order reflective layer activities. For example, the second-order reflective layer may symbolize the existence of a plan that later results in successful execution as a positive first-order reflective thinking goal. Similarly, the second-order reflective layer may symbolize the existence of a plan that later results in failure as a negative first-order reflective thinking goal. It is important to understand that first-order thinking activities are not limited to only making a specific type of plan, such as a plan that is expected to result in a successful execution.

To keep the modelling process from becoming ungrounded and circular, it is necessary to allow first-order reflective thinking to remain undefined as pre-reflective activities in Duration. The utility in this type of thinking appears necessarily because it allows for the creation of some worst possible plan so that other plans may be compared and made as different as possible from this worst case scenario. There are many creative and potentially useful planning techniques that have been described in the AI literature. Many of these techniques are based on analogical abstractions that have reasonable explanations of utility only in retrospect by reflectively creating a causal hypothesis in which the result is the symbolic goal of the existence of a successful plan.

My reflective model does not limit first-order planning activities to being subsequently recognized by the second-order reflective layer as one symbolic type or another. This emphasizes the ability of the second-order reflective layer to create causal hypotheses and create first-order reflective thinking goals that inform the second-order creation of plans for first-order thinking resources to create more successful plans for accomplishing symbolized physical goals.

6 PROBLEMATIC INTERPRETATIONS

6.1 THE MISTAKE OF SUBJECTIVE IDEALISM

Readers might confuse my model of mind with the philosophy of Subjective Idealism or Immaterialism (Berkeley 1734), since both use the term "mind" to refer to everything that exists. However, it would be a mistake to confuse this semantic detail in my explanation of my model of mind being equivalent to Subjective Idealism. The philosophy of Subjective Idealism states that everything is axiomatically divided into subject and object parts. Then, subsequently, the subject part is defined to be everything. In other words, Subjective Idealists first believe that the world is divided into two parts, the subject and objects, and then subsequently, and illogically claim, that the subject is all that exists. Obviously, there is a tautological cycle in this model. The mistake can be found in Subjective Idealism's definition of "everything". Everything functions as the origin of the model as well as the "part". There is a causal circularity in even attempting to imagine this everything that is being divided into these parts in the first place.

For example, here's another way to realize the contradiction. If one is aware of having a subjective perspective on objects, one must have a vantage point from which to be aware of this. This vantage point is necessarily outside of the subjective perspective. If the mind is the subjective perspective, then one could never become aware of being subjective, since there would be nothing in existence to contrast "everything". My theory explicitly avoids a subjective perspective on objects. Therefore, when I define the mind to be everything that exists, this limitation avoids the flaw that would prevent the modelling of an awareness of subjective points of view.

7 RELATED MODELS

7.1 THE EMOTION MACHINE

Minsky (2006) describes a six-layer reflective model of mind, *Model-6: The Emotion Machine*. Minsky's model reflects over a *physical simulation* that provides an interface to an external world with sensors and actuators for controlling a physical body. The physical simulation of Model-6 corresponds to activities in the pre-symbolic physical or reflective^0 layer of my model. Model-6 has two reactive layers, built-in and learned, that are parts of the reflective^1 layer of my model, organizing built-in and compiled plan resources, respectively. The "deliberative" layer of Minsky's model also maps to the first-order reflective thinking layer in my model, but this is only very approximate because Minsky's deliberative layer is not limited to reasoning about causal models of the physical simulation as in my model. In both of our models, this layer makes plans using causal models that include references to physical activities. In my model, I have placed an emphasis on a recursive definition of the reflective^1 layer of my model. In terms of Model-6, the built-in reactive, learned reactive, and deliberative layers are designed to be recursively applied to themselves, resulting in n duplications of each of these layers. Minsky's reflective layer maps to the union of all of the reflective layers in my model that are equal to or greater than order two. In both models, this layer responds to failures in planning or plan execution, calling upon higher orders of causal models to guide a debugging response.

$$\text{Minsky's Physical Simulation} \approx \text{reflective}^0$$

$$\left.\begin{array}{l}\text{Minsky's } n^{\text{th}} \text{ Built-in Reactive Layer}\\ \text{Minsky's } n^{\text{th}} \text{ Learned Reactive Layer}\\ \text{Minsky's } n^{\text{th}} \text{ Deliberative Layer}\end{array}\right\} \approx \text{reflective}^n \text{ for } n \geqslant 1$$

$$\text{Minsky's Reflective Layer} \approx \bigcup_{n=2}^{\infty} \text{reflective}^n$$

Figure 10: The lower layers of Model-6 mapped to the suggested reflective^n order notation.

Figure 10 shows how Minsky's layers map to the reflective^n order notation. Despite the use of set theoretic mathematical notation, I do not mean to imply that the activities that can be thought of as layers are actually sets of anything specific, especially not symbols, since the physical layer refers to activities, which are not symbolic or discrete or separate or even static by definition. The set theoretic notation is a useful tool for thinking, as long as the implication is not a strict adherence to the mathematical definitions, such as, the derivation of sets by the potentially infinite enumeration of discrete elements, which would obviously be absurd. The set notation is used only to show a picture of how to think about the mapping between the models and not to formally prove anything about the relationship between our models by using the rules of mathematical set theory.

Above Minsky's reflective layer are the "self-reflective" and "self-conscious" layers of reflective thinking. I have not implemented these layers because they are not simple extensions upward in my model. I describe how I see these layers being implemented in section 20.5 as future research.

7.2 INTERIOR GROUNDING

Minsky (2005) describes an evolutionary reflective model of mind that he calls *interior grounding*. Interior grounding states that each layer of reflective thinking could be genetically predestined to each have different and specific types of useful ways of thinking. Minsky rejects the *physical grounding hypothesis*, which stipulates that thoughts must necessarily develop from the lowest layer first and only subsequently to the higher layers of thinking.

In terms of my model, activities in the mind can create and manipulate symbolic arrangements without these symbols necessarily referring to the physical layer of activity. My model does not conflict with modelling different developmental stages of minds or genetic differences between minds; the implementation begins the simulation in a specific physical state, where the reflective layers of the mind are already all actively thinking with explicit sets of pre-existing knowledge, such as plans. My model learns new knowledge and debugs the knowledge it already has as the simulation proceeds.

In terms of Model-6, each reflective layer of my model can be considered to have a unique built-in reactive layer, which makes room for these genetic dispositions that Minsky describes as the basis for interior grounding. Also, my use of the term grounding is consistent in the sense that dynamic activity is the grounding for symbolic references in each layer of activity of my model, so this grounding need not begin by reasoning about physical activity, growing from lower layers to higher layers, but may instead begin reasoning about any layer at any given height in the model.

7.3 METAREASONING

Cox & Raja (2008) present a reflective model of mind that they refer to as *metareasoning*. The metareasoning model they present begins with a "ground level", which corresponds to the physical, reflective^0, layer, then describe how a second level, the "object level", is a control loop that receives perceptions from the ground level, processes these, and sends actionable commands back to the ground level. Their object level corresponds with the first-order reflective layer. They then proceed to a third level, which completes two cascaded control loops: one controlling the ground level with another controlling the object level. This third level is called the "meta-level". Cox and Raja's meta-level corresponds to the second-order reflective layer, but Cox and Raja's model might be interpretted to limit the meta-level to thinking about the object level, which is not true of the reflective^2 layer in the sense that this layer can symbolize any of the activities in the layers below it. Figure 11 shows how Cox and Raja's levels map

$$\begin{aligned}
\text{Cox and Raja's Ground Level} &\approx \text{reflective}^0 \\
\text{Cox and Raja's Object Level} &\approx \text{reflective}^1 \\
\text{Cox and Raja's Meta-Level} &\approx \text{reflective}^2 \\
\text{Cox and Raja's Meta}^2\text{-Level} &\approx \text{reflective}^3 \\
\text{Cox and Raja's Meta}^n\text{-Level} &\approx \text{reflective}^{(n+1)}
\end{aligned}$$

Figure 11: The lower levels of Cox and Raja's metareasoning model mapped to the suggested reflective^n order notation.

to the suggested reflective^n order notation. Here they explain what happens when the object level fails in either the creation or execution of a plan:

> When reasoning fails at some task, it may involve the explanation of the causal contributions of failure and the diagnosis of the object- level reasoning process.

Cox and Raja explain how debugging causal models of the reasoning process itself is a form of meta-level reasoning, or second-order reflective thinking. My model has a separate distinct class of causal model for each layer of reflective thinking, so learning physical causal models is a distinct activity from learning first-order reflective thinking causal models. The implementation demonstrates how two distinct layers of causal models allow not only learning to act but also learning to think about acting.

7.4 propagators

Radul & Sussman (2009) describe an object called a *cell* that is closely related to the knowledge dependencies depicted in Figure 5. These cells can be connected by dependency sets that specify functions that derive new knowledge from the dependencies, as well as, functions for merging the collected knowledge in cells. Collections of Radul and Sussman's cells are called a *propa-*

gator. The knowledge dependencies in this model can be thought of as a type of propagator. The combination of a propagator model of knowledge maintenance with a reflective organization is described as future research in chapter 20.

Part II

THE SIMULATION

8 INTRODUCING SIMULATION

Keeping distinctions between the continuous dynamic referent and discrete static symbols, a model can be constructed which allows n layers of reflection to be described. Non-reflective AI models learn by correcting one immediate cause of a failure; however, given a layered model, New opportunities for considering one failure to also be failures in the reflective layers "above" and "below" the original failure can be described, resulting in an arbitrary number of reflective learning opportunities from a single failure.

I will not attempt to give a complete representation for a general problem domain nor a complete description of a general problem solver. My work here leaves the unmodellable as known to be just that. Thinking as a dynamic activity uses models as useful tools with the clear understanding that these models are static. Further, and critically important for the advancement of the field of AI, when describing reflective thinking, it is imperative to have an explicit awareness of thinking being limited to manipulating the static. I will therefore refer to an AI model as simply an "AI" with the shared understanding that I am referring to a static model.

The simulation model is an extension of the model of mind that includes a mathematical description of a discrete "state" of the model in a discrete time. Neither the model of mind nor the simulation model changes over time. The simulation model therefore makes reference to a discretely stepped state that is used to simulate the necessarily separate, static model of mind. The term "simulation" is thus used as a general reference to the dynamic activity in Duration that manipulates and steps the state of the simulation model. In describing the simulation model, it is important to not confuse the dynamic activity of simulation with the model or the state of the model, which are both static at any given point in time. The term "simulation" is the only reference to activities in Duration. This distinction is necessary for the simulation model to not be limited to simulating a specific kind of activity.

For example, consider a simulation of topological proof. A mathematician can "simulate" the rules of topological proof by first seeing a static arrangement of symbols on paper. These symbols can be manipulated in the process of proving or disproving the initial topological statement. There is no existing logical or mathematical formulation of the activities of topological proof. I describe how a symbolic state space can be added to the model of mind for the purposes of simulation. How exactly this simulation is done is left until the next part, where the activity of simulation is assumed to be computational.

The activities in Duration in the model of mind exist prior to their symbolization, and since it is obviously not possible to refer to something prior to symbolizing it, the assumption that these activities have already been symbolized is necessary in order to mathematically describe the state of the simulation model. Defining the simulation model requires a description in terms of symbols for the "undescribed activities in Duration." Each assumption made in symbolizing the ongoing activities in Duration restricts the simulation model from being a model of those assumptions because only the symbolic result of those assumptions is available for the simulation of the activity of reflection.

9 GRAPH NOTATION

9.1 DYNAMIC ACTIVITIES AS A SET

I will begin with one of the simplest mathematical models, a *set* of symbols, to model of the activities in Duration. I will first describe the simulation model as the mathematical set, X, the activities of the state. Here is an example of a possible symbolization of the activities shown in Figure 12:

$$X = \left\{ \begin{array}{l} \texttt{Gripper-1,} \\ \texttt{Block-1,} \\ \texttt{Block-2,} \\ \texttt{Table-1} \\ \texttt{sitting-on,} \\ \texttt{being-above,} \\ \texttt{moving,} \\ \texttt{left} \end{array} \right\} \tag{9.1}$$

9.2 A GRAPH REPRESENTATION

The activities in Duration exist in a continuous Spatial arrangement that is symbolized and ordered by the reflective thinking layers. In order to simulate this continuous homogenous Space, a discrete static representation must be included in the state of the simulation model. A graph of labelled nodes and edges can be used to represent the activity of a Spatial relationship in the simulation. Because the graph provides a clear notation for referring to types of Spatial arrangements of activities, a modified notation originally from Messmer (1995) is used to define a labelled graph in Definition 9.1 and a labelled subgraph in Definition 9.2.

Definition 9.1. A graph G is the 4-tuple (V, E, μ, ν), where

- V is the set of vertices,
- $E \subseteq V \times V$ is the set of edges,
- $\mu : V \mapsto \{\ell_V\}$ is a function assigning a set of labels to each vertex.
- $\nu : E \mapsto \{\ell_E\}$ is a function assigning a set of labels to each edge.

In this definition, the edges are directed, i.e. there is an edge from v_1 to v_2 if $(v_1, v_2) \in E$. The empty graph, i.e. the graph with an empty set of vertices will be denoted by $\emptyset$. The union of the set

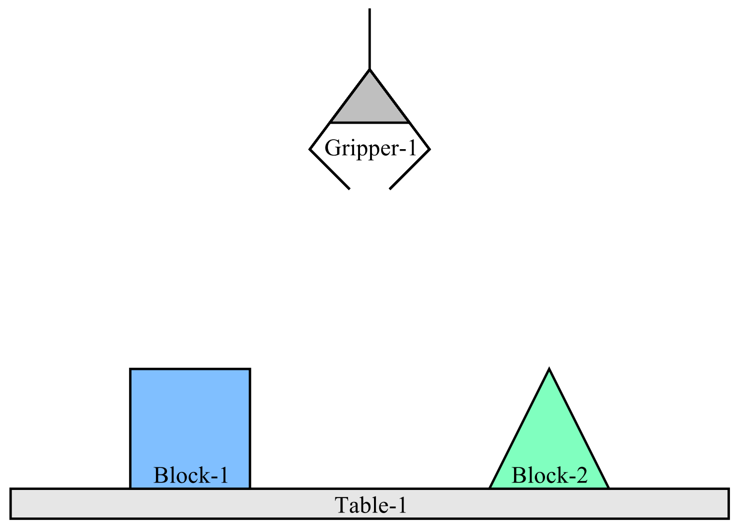


Figure 12: An example for simulation.

of labels referred to by μ and ν will be sometimes be referred to with a dot notation, $\mathring{G} = \ell_V \cup \ell_E$.

Definition 9.2. Given a graph $G = (V, E, \mu, \nu)$, a *subgraph* of G is a graph $G_s = (V_s, E_s, \mu_s, \nu_s)$ such that

1. $V_s \subseteq V$
2. $E_s \subseteq V_s \times V_s$,
3. μ_s is the restriction of μ, i.e.
$$\mu_s(e) \subseteq \begin{cases} \mu(\nu) & \text{if } \nu \in V_s \\ \text{undefined} & \text{otherwise} \end{cases}$$
4. ν_s is the restriction of ν, i.e.
$$\nu_s(e) \subseteq \begin{cases} \nu(e) & \text{if } e \in E_s \\ \text{undefined} & \text{otherwise} \end{cases}$$

From this definition it is easy to see that, given a graph G, any subset of its vertices and an included subset of its edges uniquely defines a subgraph of G. The notation $G_s \subseteq G$ is used to indicate that G_s is a subgraph of G.

9.3 GRAPH ISOMORPHISM

Definition 9.3. A bijective function $f : V \mapsto V'$ is a *graph isomorphism* from a graph $G = \{V, E, \mu, \nu\}$ to a graph $G' = \{V', E', \mu', \nu'\}$ if:

1. $\mu(\nu) = \mu'(f(\nu))$ for all $\nu \in V$.
2. For any edge $e = (\nu_1, \nu_2) \in E$ there exists an edge $e' = (f(\nu_1), f(\nu_2)) \in E'$ such that $\nu(e) = \nu'(e')$, and for any $e' = (\nu'_1, \nu'_2) \in E'$ there exists an edge $e = (f^{-1}(\nu'_1), f^{-1}(\nu'_2)) \in E$ such that $\nu'(e') = \nu(e)$.

9.4 representing continuous space as a graph

Figure 13 shows an example of the simulation state state represented as a graph. Graph notation will be used to describe the activities in Duration and continuous Space, which exist before they are symbolized and discretely ordered by the reflective[1] layer. The simulation model state space, S, is defined to be a graph in Equations 9.2 through 9.7.

$$S = (S_V,\ S_E,\ S_\mu,\ S_\nu) \tag{9.2}$$

$$S_V = \{v_1,\ v_2,\ v_3,\ v_4,\ v_5,\ v_6,\ v_7,\ v_8,\ v_9\} \tag{9.3}$$

$$S_V = \left\{ \begin{array}{l} \texttt{Gripper, Block, Table,} \\ \texttt{gripper-1, block-1, gripper-1,} \\ \texttt{table-1, left} \end{array} \right\} \tag{9.4}$$

$$S_E = \left\{ \begin{array}{l} (v_1,v_5),\ (v_1,v_4),\ (v_1,v_9),\ (v_2,v_4), \\ (v_2,v_6),\ (v_3,v_4),\ (v_3,v_7),\ (v_4,v_8) \end{array} \right\} \tag{9.5}$$

$$S_\mu(v) = \begin{cases} \{\texttt{Gripper}\} & \text{if } v = v_1 \\ \{\texttt{Block}\} & \text{if } v = v_2 \ \vee \\ & \quad v = v_3 \\ \{\texttt{Table}\} & \text{if } v = v_4 \\ \{\texttt{gripper-1}\} & \text{if } v = v_5 \\ \{\texttt{block-1}\} & \text{if } v = v_6 \\ \{\texttt{block-2}\} & \text{if } v = v_7 \\ \{\texttt{table-1}\} & \text{if } v = v_8 \\ \{\texttt{left}\} & \text{if } v = v_9 \\ \text{undefined} & \text{otherwise} \end{cases} \tag{9.6}$$

$$S_\nu(e) = \begin{cases} \{\texttt{name}\} & \text{if } e = (v_1,v_5) \ \vee \\ & \text{if } e = (v_2,v_6) \ \vee \\ & \text{if } e = (v_3,v_7) \ \vee \\ & \quad e = (v_4,v_8) \\ \{\texttt{being-above}\} & \text{if } e = (v_1,v_4) \\ \{\texttt{moving}\} & \text{if } e = (v_1,v_9) \\ \{\texttt{sitting-on}\} & \text{if } e = (v_2,v_4) \ \vee \\ & \quad e = (v_3,v_4) \\ \text{undefined} & \text{otherwise} \end{cases} \tag{9.7}$$

9.5 dot notation for graph-based frame objects

Sometimes it is useful to refer to parts of a graph by considering nodes in the graph to represent frame objects that have slotted relationships with other frame objects. Thus, I use a dot notation

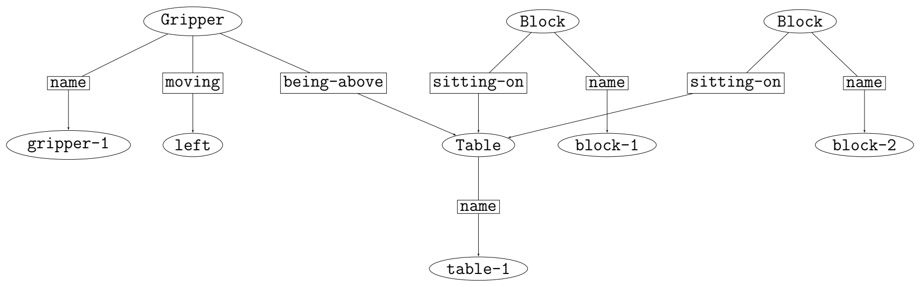


Figure 13: A labelled graph representation of a state space, where circles represent nodes, and squares represent edges.

to refer to the set of nodes that are connected to a given node, ν_1, through a given edge label, ℓ_e, as defined in Equation 9.8.

$$\nu_1.\ell_e = \{\nu_2 \ : \ (\nu_1, \nu_2) \in S_E \wedge \ell_e \in \nu[(\nu_1, \nu_2)]\} \tag{9.8}$$

9.6 SIMULATED EXISTENCE

Activities in continuous Space actually exist. Once we have considered the activities in Duration to be the set of symbols, S, we have assumed static representations for the actual dynamic. The dynamic does not have terms to help us in representing. Therefore, the question of existence becomes a question of whether or not a symbol is in the set that is the current state of the simulation, S. Equation 9.9 shows a definition of `exists`, defined as the subgraph relationship between any state, G, represented as a labelled graph, and the currently simulated state graph, S:

$$\text{exists}(G) \longleftrightarrow (G \subseteq S) \tag{9.9}$$

10 SYMBOLIC REPRESENTATION

10.1 REPRESENTING REFLECTIVE THINKING ACTIVITY

Reflective thinking exists as dynamic activities in Duration alongside the physical activities that have just been described. I will refer to the subgraph of the simulation state, S, that represents the reflective^1 thinking activities as $\mathcal{R}^1$. The first symbolic references to reflective thinking are now introduced to this layer of the model. Note that I have not introduced any specific symbolic physical activity to the model. The block example is just that, an example. Having the reflective focus be undescribed is important because then the focus is not restricted, which allows a recursive application of reflection to itself, whatever the specific graph representation described. This is a key point because the general model of reflection applying to *any* given labelled graph means that it allows for this recursive definition of layered reflective thinking.

10.2 PHYSICAL ACTIVITY IS ANY PRIOR EXISTING GRAPH

The graph of physical activities must necessarily exist prior to reflectively thinking about it. Because this prior activity is defined to be *any* graph, what remains in order to build n layers of reflective thinking activity is to define planning activity that reflects upon this prior activity. Therefore, subsequently, no matter what the reflective thinking description, this reflective thinking activity can be duplicated in n reflective layers that will think about any given graph representation of activity. Allowing the physical activity to a be an undescribed graph is important in order to allow this recursion. Notice that the physical activity is not necessarily composed of anything resembling objects, since graphs are a general representation that could just as easily represent one single number line, or even an infinitely finely interpolated multidimensional Space. If some prior physical activity can be represented as a graph, then this simulation of reflective thinking is applicable.

10.3 DEFINING N LAYERS OF REFLECTIVE THINKING ACTIVITIES

Layers of reflective activity are defined inductively, beginning with reflective^0 layer of activity, $\mathcal{R}^0$, as the initial given physical activity. Then, each subsequent layer of reflective^{n+1} thinking activity, $\mathcal{R}^{n+1}$, is defined in terms of $\mathcal{R}^{n}$ in Equations 10.1 through 10.5.

$$\mathcal{R}^{0} \equiv \textit{Physical Layer of Activities} \tag{10.1}$$

$$\mathcal{R}_V^{n+1} \subset S_V \setminus \bigcup_{k=0}^{n} \mathcal{R}_V^{k} \tag{10.2}$$

$$\mathcal{R}_E^{n+1} = (\mathcal{R}_V^{n+1} \times \mathcal{R}_V^{n+1}) \cap S_E \tag{10.3}$$

$$\mathcal{R}_\mu^{n+1}(\nu) = \begin{cases} S_\mu(\nu) & \text{if } \nu \in \mathcal{R}_V^{n+1} \\ \text{undefined} & \text{otherwise} \end{cases} \tag{10.4}$$

$$\mathcal{R}_\nu^{n+1}(e) = \begin{cases} S_\nu(e) & \text{if } e \in \mathcal{R}_E^{n+1} \\ \text{undefined} & \text{otherwise} \end{cases} \tag{10.5}$$

Note that there can be edges between activities in subsequent layers. These edges are not within any of the graphs of the reflective layers of thinking, but they do exist in the state graph, S. The edges that relate the reflective layers of activity in the simulation model are a necessary component of dynamic symbolic references to activities in the layers below the symbolic activity. In other words, $\bigcup_{k=0}^{\infty} \mathcal{R}^{k} \subset S$, the simulation state is greater than the union of its reflective layers.

10.4 REPRESENTING STATIC SYMBOLS

In order to describe symbolization, a representation for a simulated symbol must be defined. A symbol is the most interesting object in the model because it functions as a static reference to the dynamic, while being fundamentally dynamic itself. There are two primary features of a symbolic reference:

- A symbol functions as a static reference and, thus, can be ordered in Spatial arrangements that function as static orderings.
- A symbol is fundamentally dynamic and, thus, exists as dynamic activities in a referential dynamic continuous Spatial relationship with other dynamic activities in Duration.

A symbol is actively maintained in Duration, so the existence of a simulated symbol in the simulation model would need to be added to the set of all simulated activities in Duration. The set of symbols* in layer n is defined to be X^{n*}. Equations 10.6 and 10.7 define layered sets of symbols*:

$$X^{n*} = \left\{ x^* : \begin{array}{c} x^* \in \mathcal{R}_V^{n} \ \wedge \\ \left(\forall_{\nu \in \mathcal{R}_V^{n}}, \begin{array}{c} S_\mu(\nu) = \texttt{Reflective} \rightarrow \\ x^* \in \nu.\texttt{symbol} \end{array} \right) \end{array} \right\} \tag{10.6}$$

$$x^{n*} \in X^{n*} \tag{10.7}$$

10.5 REPRESENTING SYMBOLIC REFERENCE

Because dynamic activities in Duration cannot be ordered in Space by the reflective thinking layers, the most important aspect of the reflective simulation is the capability to create a static reference, x^*, which has an associated dynamic representation of an arbitrary subgraph of activity, $\Psi(x^*)$. Note that while $\Psi(x^*)$ represents a potential subgraph of the simulation state, this subgraph may or may not actually exist in the simulation state, S. This symbolic representative capability is the basis of symbolic perception and memory.

Because the simulation state is a static symbolic representation of the actual dynamic activities in Duration, simulation of the dynamic reflective representation must also be in the same simulated terms of dynamic physical activities. Considering this dynamic activity to be symbolic helps in explaining the relationship between the dynamic and the static, as long as the distinction is kept clear and the simulation of the dynamic is not confused with the simulation of the static. The confusion would arise because both are modelled as symbols in the state graph, S, of the simulation model. Of course, in actuality, the dynamic physical activities do not have symbolic terms. I will continue to use the asterisk (*) notation when referring to the simulation of actively static symbolic references as opposed to the simulation of the otherwise purely dynamic activities in Duration. Equations 10.8 through 10.13 show a definition of a static symbolic reference, x^*, to a dynamic reflective representation of a subgraph of activity, $\Psi(x^*)$, not necessarily actually within the state graph, S.

$$\forall_{v \in \Psi_V(x^*)},\ v \in x^*.\mathtt{node} \tag{10.8}$$

$$\forall_{v \in \Psi_V(x^*)},\ v.\mathtt{label} = \{\Psi_\mu(x^*)(v)\} \tag{10.9}$$

$$\forall_{e \in \Psi_E(x^*)},\ e \in x^*.\mathtt{edge} \tag{10.10}$$

$$\forall_{e \in \Psi_E(x^*)},\ [e = (e_{v_1}, e_{v_2}) \wedge e.\mathtt{tail} = \{e_{v_1}\}] \tag{10.11}$$

$$\forall_{e \in \Psi_E(x^*)},\ [e = (e_{v_1}, e_{v_2}) \wedge e.\mathtt{head} = \{e_{v_2}\}] \tag{10.12}$$

$$\forall_{e \in \Psi_E(x^*)},\ e.\mathtt{label} = \{\Psi_\nu(x^*)(e)\} \tag{10.13}$$

Figure 14 shows an example of the representation for a symbolic reference to the reflective^0 layer in the reflective^1 layer.

10.6 A VISUALIZATION OF A REFLECTIVE RELATIONSHIP

When reflective graph structures, as in Figure 14, get to be larger, they can be confusing when shown in full structural detail. In order to simplify the visual representation of a reflective relationship, I will sometimes use a trapezoidal edge-like visual in order to present the same complicated relationship structure in a simpler way. Figure 15 shows a simpler visualization of the same example reflective relationship.

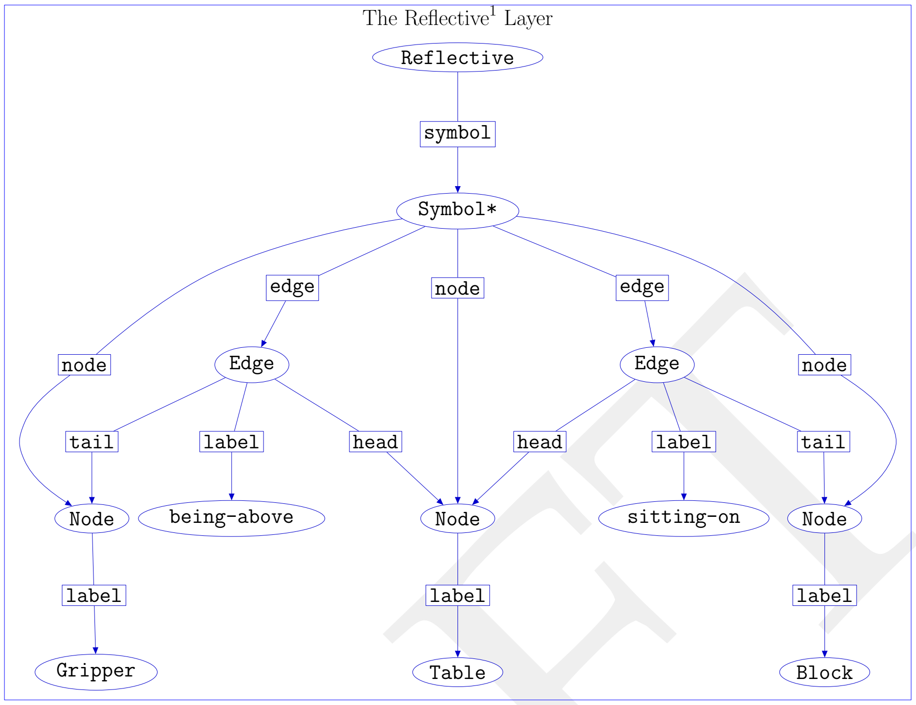


Figure 14: An example of the representation for symbolic reference, where the dynamic referent is the physical activity of a block being on a table in conjunction with a gripper being above the same table.

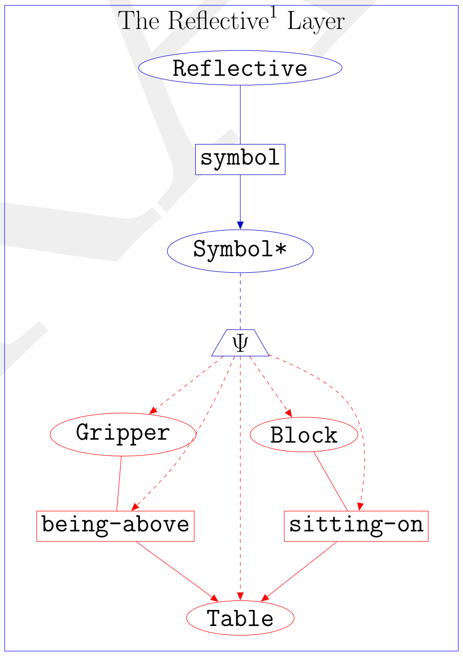


Figure 15: A shorthand visualization for a symbolic reference, where the trapezoid is a shorthand, for the same representation depicted in Figure 14. The symbolic referent subgraph is pictured in red because this subgraph does not actually exist in the simulation state graph, S. Instead, the symbolic referent subgraph's *representation* actually dynamically exists in the simulation state graph, S. Thus, the symbolic representation operator, Ψ, is not a symbol in the state graph, S, but represents the relationship to the representation of the non-existent subgraph that appears in red.

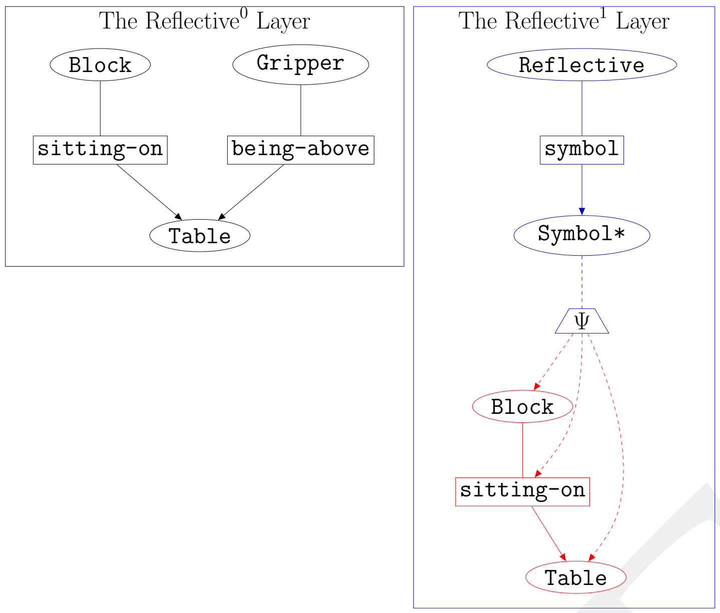


Figure 16: Example of a perceived first-order symbolic reference to the physical layer of activity, i.e. $\Psi(x^*)$ is isomorphic to a subgraph of $\mathcal{R}^0$.

10.7 REPRESENTING SYMBOLIC PERCEPTION AND MEMORY

Figure 16 shows an example of a symbol*, x^*, in the first-order reflective layer. Notice that $\Psi(x^*)$ is isomorphic to a subgraph of the reflective^0 layer, $\mathcal{R}^0$. When a symbolic representation is isomorphic to a subgraph of the layers below it, the symbol is said to be a *symbolic perception*, or simply, a *percept*. Equation 10.14 defines when a symbolic referent is considered a percept based on its isomorphic relationship with layers of activity below it:

$$\text{percept}^n(x^*) = \begin{cases} \textit{True} & \text{if } \Psi(x^*) \text{ is isomorphic to a} \\ & \quad \text{subgraph of } S \setminus \bigcup_{k=n}^{\infty} \mathcal{R}^k \\ \textit{False} & \text{otherwise} \end{cases} \tag{10.14}$$

Figure 17 shows a symbolic reflective reference to the physical layer of activity that is not perceived because it is not isomorphic to any subgraph in the reflective^0 layer, $\mathcal{R}^0$. When a symbol is not isomorphic to a subgraph in the layers below it, this is referred to as a *symbolic memory*.

10.8 SECOND-ORDER SYMBOLIC REPRESENTATION

The second-order layer of reflective thinking is the first layer that can reflect on the existence of the symbols of the first-order reflective layer. Figure 18 shows an example of a second-order symbolic reference to a first-order symbolic representation.

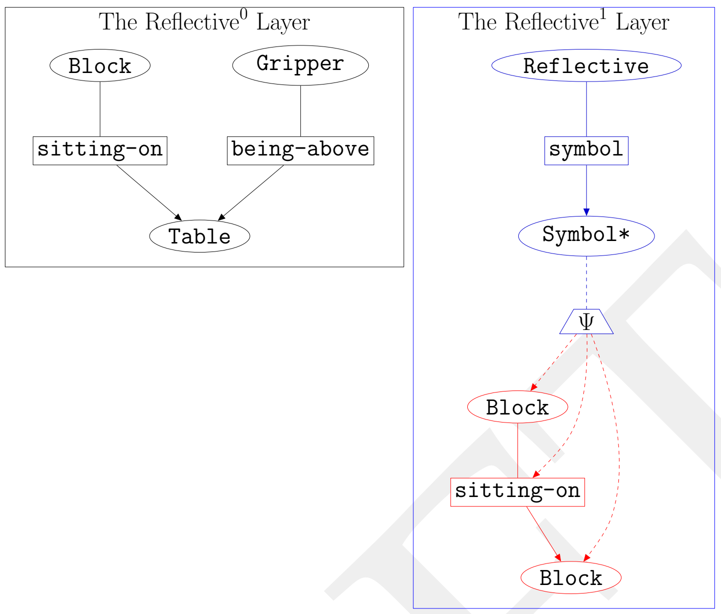


Figure 17: Example of a non-perceived symbol, or symbolic memory, x^*, i.e. $\Psi(x^*)$ is not isomorphic to a subgraph of $\mathcal{R}^0$.

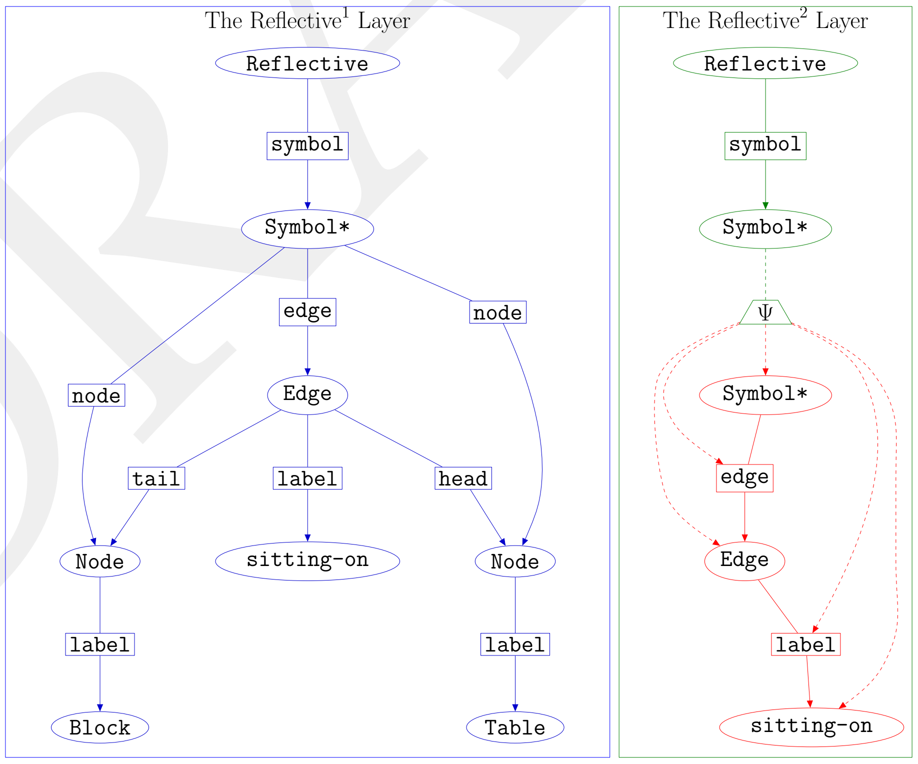


Figure 18: Example of a second-order symbol representing an aspect of first-order symbolization. Note that the actually existent symbolic representation in the first-order reflective layer is shown in full detail, while the red subgraph in the second-order symbolic representation is short-hand for the much more complex actually existent activity in this layer.

10.9 REPRESENTING SYMBOLIC RESOURCES

Each reflective thinking layer in the model has a set of built-in resources. These built-in resources correspond to what Minsky (2006) refers to as the built-in reactive resources. Resources can be in activated or suppressed Spatial arrangements. These activated and suppressed relationships can lead to logical conflicts, a type of activation failure.

Because the simulation is of actual dynamic reflective thinking and not of a simulation of an already symbolic control system, how to think about resources in my system appears as something like a cooperation between the physical layer and the first-order reflective resources. The set of resources, A^{n*}, is a subset of the symbols, X^{n*}, in any given layer. They are symbolic references to the lower layers of continuous dynamic activity. Figure 19 shows an example of a first-order reflective resource, a symbolic reference to the reflective^0 layer of dynamic physical activity. Equation 10.15 through 10.18 define a symbolic resource*, a^{n*}, in the simulation state graph, S.

$$A^{n*} \equiv \textit{Reflective}^n \textit{ action resources.} \tag{10.15}$$

$$A^{n*} \subseteq X^{n*} \tag{10.16}$$

$$\texttt{reflective}^n\texttt{.resource} = A^{n*} \tag{10.17}$$

$$a^{n*} \in A^{n*} \tag{10.18}$$

Figure 20 shows an example of four resources, each displaying a different activation or suppression Spatial relationship:

1. An example of an activation relationship.
2. An example of a suppression relationship.
3. An example of an idle resource in neither an activation nor a suppression relationship.
4. An example of a resource in simultaneous, conflicting, activation and suppression relationships.

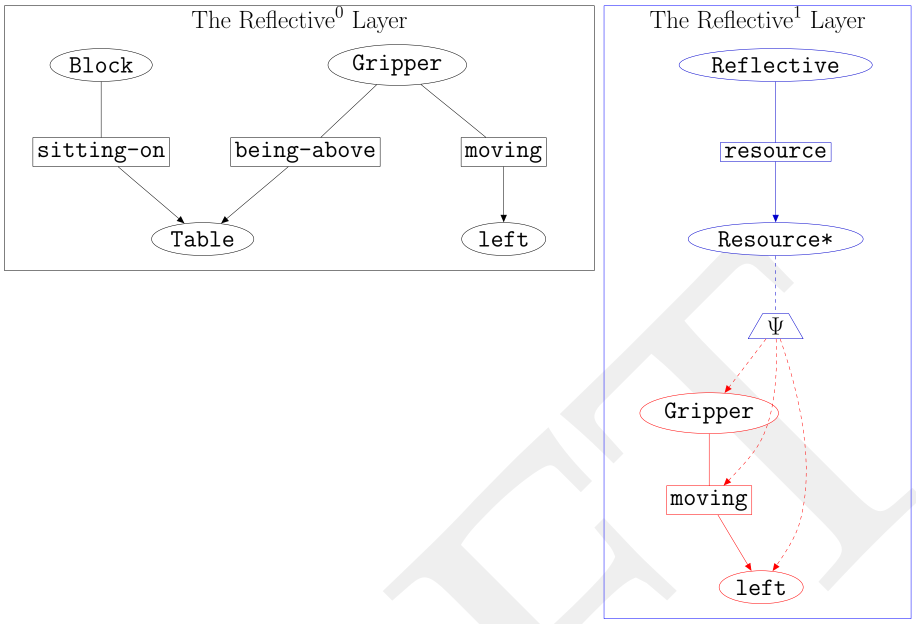


Figure 19: An example of a first-order reflective resource that refers to the dynamic activity in the reflective^0 layer of physical activity.

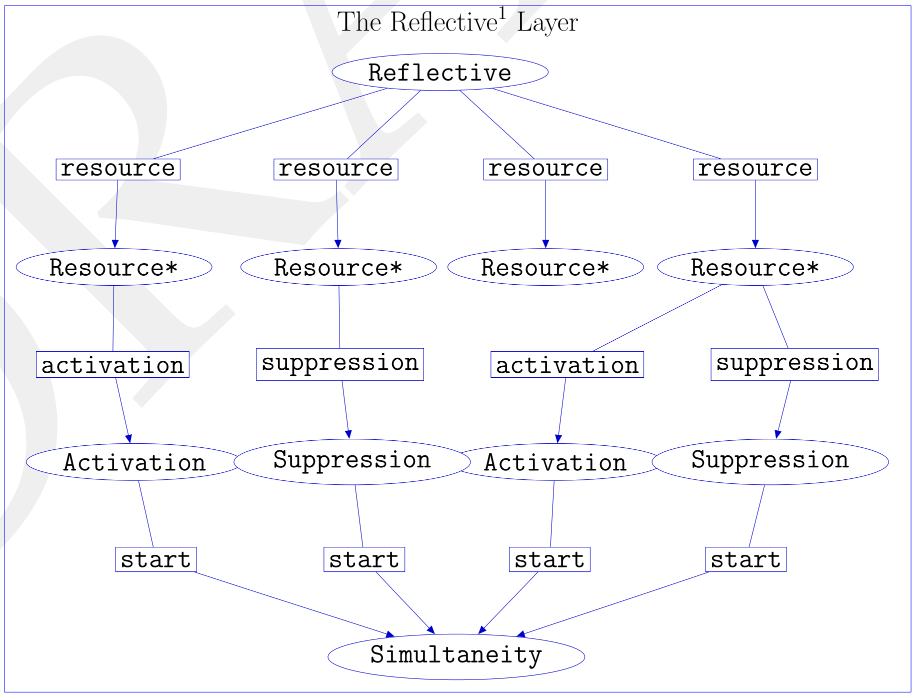


Figure 20: Four examples of resource activation and suppression Spatial relationships: (1) activation, (2) suppression, (3) idle, and (4) conflict. Note that all of these activation and suppression events have the same starting simultaneity and no ending. In general, activation and suppression events may be historical memories, and do not necessarily imply present activation, suppression, or conflict.

11 GROUNDED FACTUAL KNOWLEDGE

Spatial arrangements of static symbols are created by the reflective thinking layers. Static symbols are not contained within the physical layer, but these symbols are used to represent ongoing causal transitions from the past to the future. These grounded representations form the basis from which causal hypotheses can be abstracted. Hypothetical causal models enable planning activities to imagine counterfactual knowledge of the past and future. Second-order dynamically grounded causal transitions that factually describe this first-order planning activity allows the abstraction of second-order hypothetical causal knowledge that allows the creation of counterfactual knowledge about the hypothetical necessities and results of planning activities themselves.

This chapter will discuss how grounded knowledge representations form the basis of hypothetical abstractions of both physical activities as well as reflective thinking activities. Having a dynamically grounded reference for factual knowledge about every layer of reflective thinking is necessary for debugging hypothetical counterfactual conclusions in any layer when these conclusions turn out to be wrong. Keeping clear distinctions between dynamically grounded factual knowledge and the subsequent practically motivated hypothetical derivations, gives a clear way to think about facts that are known to be true, while the symbolic basis for this grounded factual knowledge can be reflectively refined when the utility of hypothetical derivative assumptions end up failing to be useful.

Thus, I must first describe the N layers of dynamically grounded factual causal transitions that are the basis for the hypothetical and counterfactual derivations used for the N layers of planning. Clearly understanding the derivation of N layers of planning knowledge leads naturally to a simple debugging method for N reflective layers of knowledge maintenance, the focus of this thesis.

11.1 SIMULTANEITIES

Simultaneities form the simplest component of the representational machinery of present grounded factual knowledge in the simulation model. Using the notation of the previous chapter, the set of currently perceived symbols, $\mathcal{P}^{n*}$, and the set of unperceived symbols, $\overline{\mathcal{P}}^{n*}$, in the nth-order reflective thinking layer are defined in Equations 11.1 and 11.2.

$$\mathcal{P}^{n*} = \{x^* : x^* \in X^{n*} \wedge \text{percept}^n(x^*)\} \tag{11.1}$$

$$\overline{\mathcal{P}}^{n*} = \{x^* : x^* \in X^{n*} \wedge \neg\text{percept}^n(x^*)\} \tag{11.2}$$

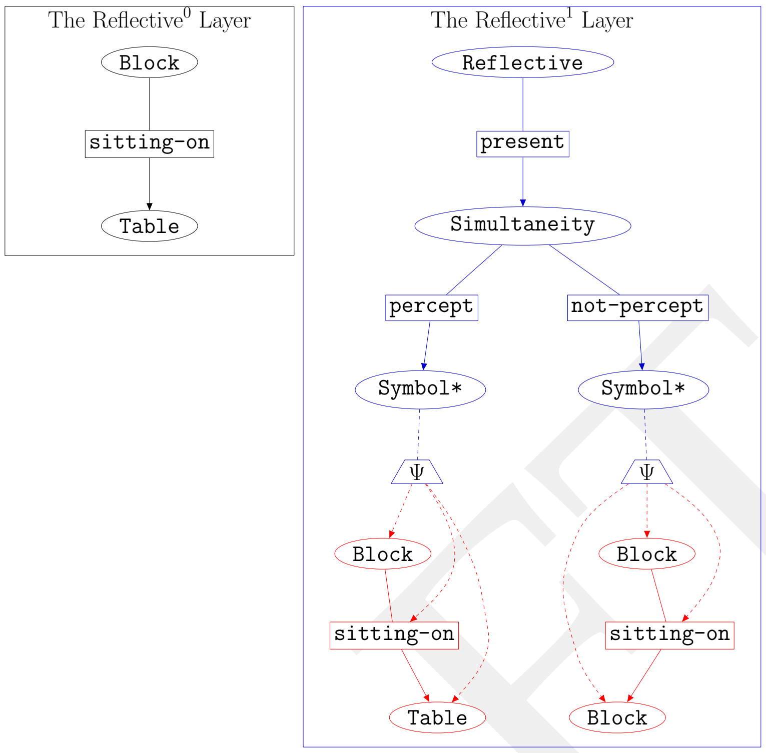


Figure 21: Example simultaneity of positive and negative symbolic perceptions, where the symbolic reference to the dynamic physical activity of a block sitting on a table is perceived, while a block sitting on another block is not.

Equation 11.3 defines the graph representation that exists in the simulation state, S, of the present grounded factual simultaneity for each reflective^n layer of thinking activity.

$$\begin{aligned} (t \in \texttt{reflective}^n.\texttt{present}) \longrightarrow \\ (t.\texttt{percept} = \mathcal{P}^{n*}) \ \wedge \\ (t.\texttt{not-percept} = \overline{\mathcal{P}}^{n*}) \end{aligned} \tag{11.3}$$

Figure 21 shows a simple example of a present grounded factual simultaneity in the first-order reflective thinking layer.

11.2 TRANSITIONS

Simultaneities can be ordered in temporal relationships in each reflective layer. Equation 11.4 defines the graph representation in the simulation state, S, that keeps track of the grounded knowledge of all temporal orderings of previous perceptual simultaneities.

$$\texttt{reflective}^n.\texttt{time} \equiv \textit{Transitions of the Reflective}^n \textit{ Layer} \tag{11.4}$$

Given this representation of temporal orderings for each reflective^n layer, Equations 11.5 and 11.6 define the transitive temporal re-

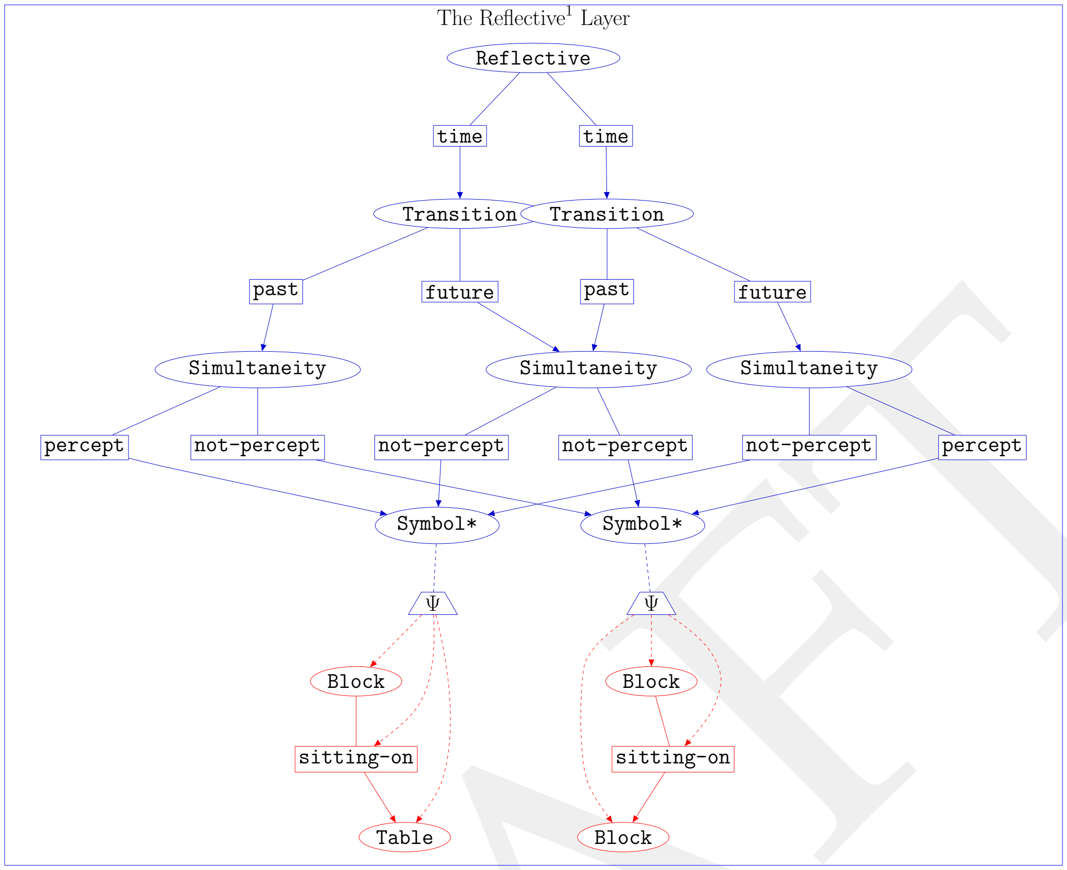


Figure 22: An example of two transitions ordering three simultaneities: (1) a block sitting on a table, (2) a block not sitting on a table, and (3) a block sitting on another block.

lationship, $\overset{n}{<}$, that describes a separate ordering of grounded simultaneities for each reflective layer.

$$\left(\begin{array}{l} x^* \in \texttt{reflective}^n.\texttt{time} \ \wedge \\ \quad t_1 \in x^*.\texttt{past} \ \wedge \ t_2 \in x^*.\texttt{future} \end{array} \right) \longrightarrow t_1 \overset{n}{<} t_2 \quad (11.5)$$

$$(t_1 \overset{n}{<} t_2 \ \wedge \ t_2 \overset{n}{<} t_3) \longrightarrow t_1 \overset{n}{<} t_3 \quad (11.6)$$

Equation 11.7 defines the separate set of ordered simultaneities within each reflective^n layer.

$$T^n = \left\{ t : \begin{array}{l} x^* \in \texttt{reflective}^n.\texttt{time} \ \wedge \\ \quad (t \in x^*.\texttt{past} \ \vee \ t \in x^*.\texttt{future}) \end{array} \right\} \quad (11.7)$$

Equation 11.8 defines each set of temporally ordered simultaneities, T^n, is a connected and fully ordered sequence.

$$(t_1 \in T^n \wedge t_2 \in T^n) \longrightarrow (t_1 \overset{n}{<} t_2 \vee t_2 \overset{n}{<} t_1 \vee t_1 = t_2) \quad (11.8)$$

Figure 22 shows an example of how three simultaneities can be represented in a simulation state graph as temporally ordered using two transition relationships.

11.3 PERCEPTUAL EVENTS

Equation 11.9 defines a “perceived” Spatial relationship for each reflective layer that can exist between a temporal simultaneity, t, and a static symbolic reference, x^*, to the layers below.

$$\text{perceived}^n(t, x^*) = (t \in T^n \ \wedge \ x^* \in \texttt{t.percept}) \tag{11.9}$$

Equation 11.10 defines the start times for perceptual events in each reflective layer. A perceptual event begins at the time when a symbol is perceived and has just immediately transitioned from not being perceived.

$$\begin{array}{l} \text{start}^n(t, x^*) = \\ \quad \begin{pmatrix} \text{perceived}^n(t, x^*) \ \wedge \\ \quad \begin{bmatrix} \forall_{t_1 \in T^n}, \\ \quad \begin{pmatrix} \begin{bmatrix} t_1 \overset{n}{<} t \ \wedge \\ \quad \text{perceived}^n(t_1, x^*) \end{bmatrix} \longrightarrow \\ \quad \exists_{t_2 \in T^n}, \\ \qquad \begin{bmatrix} t_1 \overset{n}{<} t_2 \overset{n}{<} t \ \wedge \\ \quad \neg\text{perceived}^n(t_2, x^*) \end{bmatrix} \end{pmatrix} \end{bmatrix} \end{pmatrix} \end{array} \tag{11.10}$$

Equation 11.11 defines the end times for perceptual events in each reflective layer. A perceptual event ends at the time when a symbol is perceived and will immediately transition to not being perceived at the subsequent time.

$$\begin{array}{l} \text{end}^n(t, x^*) = \\ \quad \begin{pmatrix} \text{perceived}^n(t, x^*) \ \wedge \\ \quad \begin{bmatrix} \forall_{t_1 \in T^n}, \\ \quad \begin{pmatrix} \begin{bmatrix} t \overset{n}{<} t_1 \ \wedge \\ \quad \text{perceived}^n(t_1, x^*) \end{bmatrix} \longrightarrow \\ \quad \exists_{t_2 \in T^n}, \\ \qquad \begin{bmatrix} t \overset{n}{<} t_2 \overset{n}{<} t_1 \ \wedge \\ \quad \neg\text{perceived}^n(t_2, x^*) \end{bmatrix} \end{pmatrix} \end{bmatrix} \end{pmatrix} \end{array} \tag{11.11}$$

Between the start and end times of a perceptual event, a symbol has been continuously perceived. Event knowledge is factual knowledge that is grounded in the perception of the dynamic. Factual knowledge does not involve hypothetical reasoning. Later, I will show how hypothetical assumptions can be abstracted from factual event knowledge. Grounded factual knowledge is the source of all hypothetical abstractions that are used in planning. Thus, grounded knowledge is the necessary foundation for creating a planning algorithm that can be debugged when the hypothetical bases of counterfactual knowledge turn out to be contradicted by further factual knowledge. Equations 11.12 through 11.19 define a perceptual event in the n^{th}-order reflective layer.

$$e \in \texttt{reflective}^n\texttt{.event} \tag{11.12}$$
$$e\texttt{.symbol} = \{x^*\} \tag{11.13}$$
$$e\texttt{.start} = \{t_{start}\} \tag{11.14}$$
$$e\texttt{.end} = \{t_{end}\} \tag{11.15}$$
$$t_{start} \in \{t : t \in T^n \wedge \text{start}^n(t, x^*)\} \tag{11.16}$$
$$t_{end} \in \{t : t \in T^n \wedge \text{end}^n(t, x^*)\} \tag{11.17}$$
$$t_{start} \overset{n}{<} t_{end} \tag{11.18}$$
$$\neg\exists_{t \in T^n}, [t_{start} \overset{n}{<} t \overset{n}{<} t_{end} \wedge \text{end}^n(t, x^*)] \tag{11.19}$$

11.4 event transframes

While transitions represent a temporal relationship between two simultaneities, a *transframe* represents the difference between two simultaneities. For example, two simultaneities could both contain a lot of perceptual information, but the transframe between them might be a very small set of differences. The difference between two simultaneities is kept track of as two lists: (1) the additions and (2) the removals from the actively perceived symbols in each simultaneity. Figure 23 shows an example of a perceptual event that has an associated transframe.

11.5 resource activation events

Activation events keep track of the start and end simultaneities for the activation of each resource. Activation and suppression events are similar to symbolic perceptual events, except that they are based on the activation and suppression relationships that are caused by the execution of plans rathar than the isomorphic perceptions from the layers below. Although, the cause of resource events are different from perceptual events, they still have a basis in the grounded knowledge terms of their start and end simultaneities and the transframes that are computed from these factual pieces of information. The perceptual simultaneities and transframes that are related to perceptual and resource events are factual and not based on hypotheses. Figure 24 shows an example of a resource in the first-order reflective thinking layer with a completed activation event. Causal hypotheses are derived from this type of factual resource information. It is important to keep clear distinctions between what is factual and what is hypothetical knowledge in the model. Only in this way may the hypothetical counterfactual underpinnings of knowledge be debugged when they turn out to be wrong. Equations 11.20 through 11.25 define the factual positive and negative preconditions, pre^+ and pre^-, as well as positive and negative postconditions, post^+

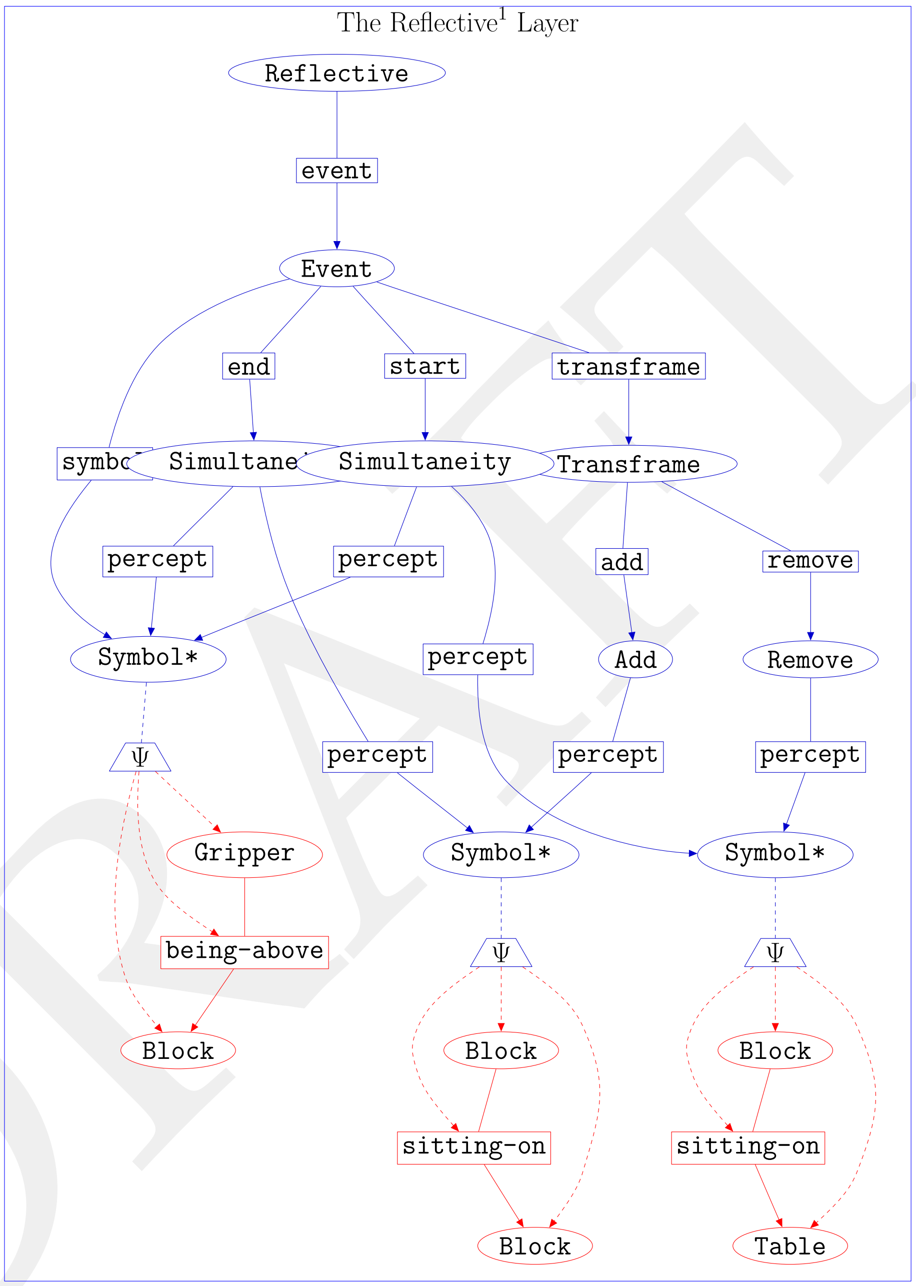


Figure 23: An example of a factual event transframe, representing the continuous perception of the symbol that refers to the dynamic physical activity of a gripper being above a block. The symbolized perceptual difference between the starting and ending simultaneities is represented by a transframe of the perceptual event, listing the symbolic additions and removals from the starting and ending situations. Note that negative perceptions in both the simultaneities and the transframe have been omitted for clarity.

and post$^-$, for each activation event, a^{n*}_ϵ, of a given resource action, a^{n*}.

$$a^{n*} \in A^{n*} \tag{11.20}$$

$$a^{n*}_\epsilon \in a^{n*}.\mathtt{activation} \tag{11.21}$$

$$\text{pre}^+(a^{n*}_\epsilon) = \left\{ x^* : \begin{array}{l} t \in a^{n*}_\epsilon.\mathtt{start} \;\wedge \\ \quad x^* \in t.\mathtt{percept} \end{array} \right\} \tag{11.22}$$

$$\text{pre}^-(a^{n*}_\epsilon) = \left\{ x^* : \begin{array}{l} t \in a^{n*}_\epsilon.\mathtt{start} \;\wedge \\ \quad x^* \in t.\mathtt{not\text{-}percept} \end{array} \right\} \tag{11.23}$$

$$\text{post}^+(a^{n*}_\epsilon) = \left\{ x^* : \begin{array}{l} t \in a^{n*}_\epsilon.\mathtt{end} \;\wedge \\ \quad x^* \in t.\mathtt{percept} \end{array} \right\} \tag{11.24}$$

$$\text{post}^-(a^{n*}_\epsilon) = \left\{ x^* : \begin{array}{l} t \in a^{n*}_\epsilon.\mathtt{end} \;\wedge \\ \quad x^* \in t.\mathtt{not\text{-}percept} \end{array} \right\} \tag{11.25}$$

$$\tag{11.26}$$

Equations 11.27 and 11.28 define the factual positive and negative transframe changes, Δ^+ and Δ^-, between the preconditions and the postconditions for each resource activation event, a^{n*}_ϵ.

$$\Delta^+(a^{n*}_\epsilon) = \text{post}^+(a^{n*}_\epsilon) \setminus \text{pre}^+(a^{n*}_\epsilon) \tag{11.27}$$

$$\Delta^-(a^{n*}_\epsilon) = \text{post}^-(a^{n*}_\epsilon) \setminus \text{pre}^-(a^{n*}_\epsilon) \tag{11.28}$$

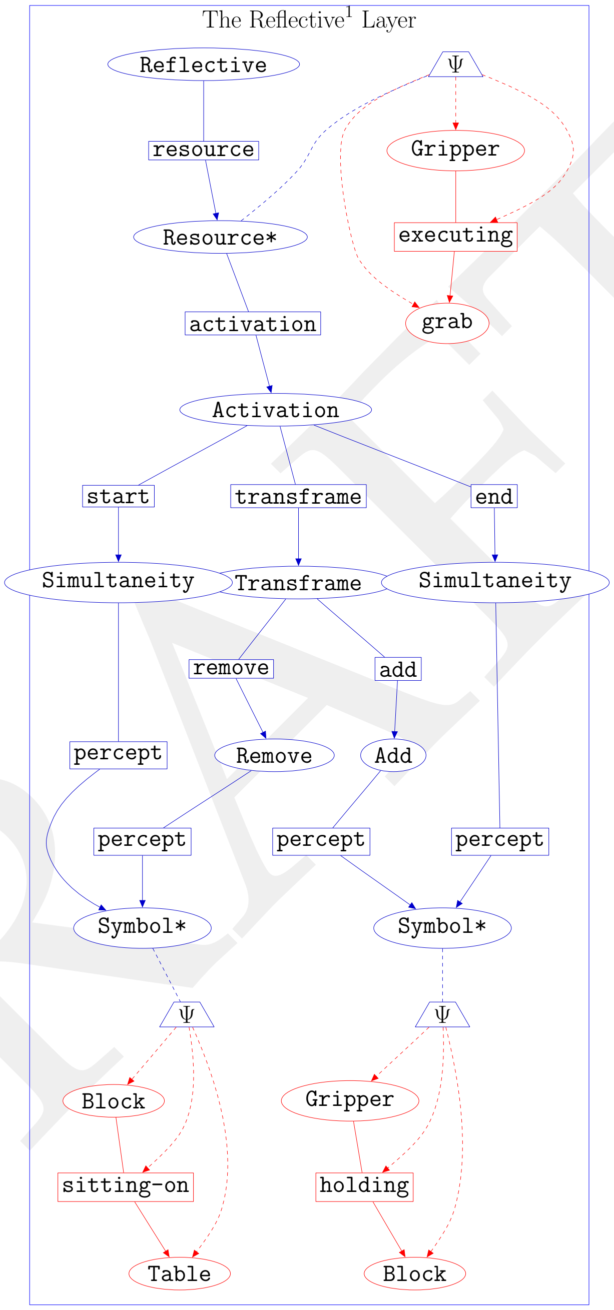


Figure 24: An example of a first-order reflective resource with an activation event, related to grounded factual start and end perceptual simultaneities as well as a grounded factual transframe. Unperceived symbols are omitted for clarity. Collections of factual resource activation events such as this are the basis of the subsequently abstracted hypothetical and counterfactual knowledge.

12 HYPOTHESES AND COUNTERFACTUAL KNOWLEDGE

12.1 HYPOTHESES

I have previously described grounded factual knowledge that has an unquestionable basis in symbolic perceptions. Now, I will describe how hypothetical causal models are abstracted from this factual knowledge in order to create counterfactual knowledge, which keeps this factual references so that it can be debugged when it is wrong.

12.2 HYPOTHESIS SPACE

Mitchell (1997) presents a general formalism for learning hypotheses from sets of ground truth examples. He describes a function approximation approach to the discrimination problem of predicting a binary value given a set of binary features. In order to keep the model of learning as general as possible, Mitchell introduces the concept of a *hypothesis space*. Given a language for describing hypotheses, the hypothesis space includes all possible expressions in this language. The hypothesis language can be arbitrarily general, but in order to demonstrate my thesis about reflective thinking, I make the simplifying assumption that this language is simply based on the logical "and" expression. This is called the *conjunctive hypothesis space*. The conjunctive hypothesis space consists of all logical conjunctions of the input features.

12.3 THE GENERAL-TO-SPECIFIC ORDERING OF HYPOTHESES

Definition 12.1. Let h_j and h_k be Boolean-valued functions defined over X. Then h_j is more-general-than-or-equal-to h_k (written $h_j \geqslant_g h_k$) if and only if

$$\forall_{x \in X}, [(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

Depending on the choice of hypothesis language, it is not always trivial to determine whether one hypothesis is more general than another. All non-redundant hypotheses in the conjunctive hypothesis space of four input variables is shown as ordered in a generalization lattice in Figure 25.

12.4 CONCEPT VERSION SPACES

Given a general-to-specific ordering of the entire hypothesis space, Mitchell describes an efficient representation of all hypotheses that match a given set of input examples, which he

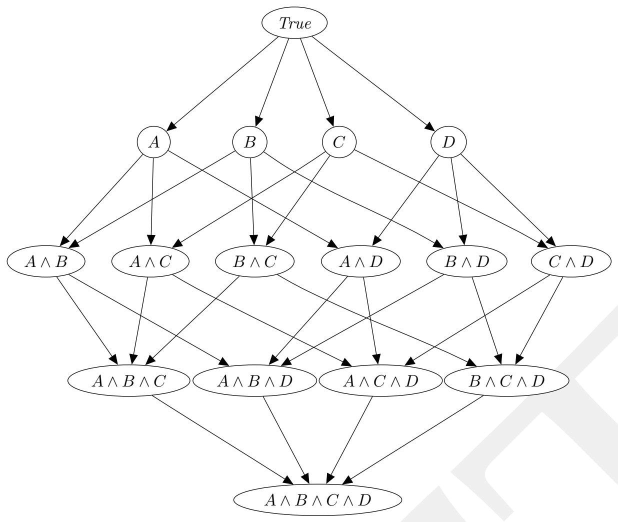


Figure 25: An example of a conjunctive hypothesis space of four logical variables, A, B, C, and D. All possible conjunctive hypotheses represented as nodes in a lattice with general-to-specific orderings represented by edges.

calls a *version space*. The implementation uses this very efficient version space representation to keep track of all hypotheses in the conjunctive hypothesis space that are consistent with factual knowledge.

12.5 HYPOTHESIZING TRANSFRAMES FROM PRECONDITIONS

In the model, planning is based on predicting the hypothetical effects of resource actions. In order to predict counterfactual simultaneities, which can fill future slots of counterfactual transitions, each historical resource activation transframe is considered as input to a hypothesis version space concept learning algorithm.

Equations 12.1 through 12.3 define the precondition symbols, a^*_E, that can be used to hypothetically predict the add and remove transframe symbols, a^*_{D+} and a^*_{D-}, for the general execution of the action resource, a^*.

$$a^*_X = \bigcup_{\epsilon \in a^*.\texttt{activation}} [\text{pre}^+(\epsilon) \cup \text{pre}^-(\epsilon)] \tag{12.1}$$

$$a^*_{D+} = \bigcup_{\epsilon \in a^*.\texttt{activation}} \Delta^+(\epsilon) \tag{12.2}$$

$$a^*_{D-} = \bigcup_{\epsilon \in a^*.\texttt{activation}} \Delta^-(\epsilon) \tag{12.3}$$

Equations 12.4 through 12.6 define the set of hypothesis spaces, $a^{n*}_{\mathcal{H}}$, for each action resource, a^{n*}.

$$a^{n*} \in A^{n*} \tag{12.4}$$

$$a^{n*}_{\mathcal{H}} \equiv \textit{Resource hypothesis spaces} \tag{12.5}$$

$$a^{n*}.\texttt{hypothesis-space} = a^{n*}_{\mathcal{H}} \tag{12.6}$$

$$\tag{12.7}$$

Equations 12.8 through 12.14 define a conjunctive hypothesis space, a^{n*}_{H}, for an action resource, a^{n*}.

$$a^{n*}_{H} \in a^{n*}_{\mathcal{H}} \tag{12.8}$$

$$a^{n*}_{H}.\texttt{add} \subseteq a^{*}_{D+} \tag{12.9}$$

$$a^{n*}_{H}.\texttt{remove} \subseteq a^{*}_{D-} \tag{12.10}$$

$$a^{n*}_{H_G} \equiv \textit{Hypothesis space general hypotheses} \tag{12.11}$$

$$a^{n*}_{H}.\texttt{general} = a^{n*}_{H_G} \tag{12.12}$$

$$a^{n*}_{H_S} \equiv \textit{Hypothesis space specific hypotheses} \tag{12.13}$$

$$a^{n*}_{H}.\texttt{specific} = a^{n*}_{H_S} \tag{12.14}$$

Figure 26 shows an example of a hypothesis space.

12.6 decisions with hypothesis dependencies

Hypothesis spaces are learned in order to predict counterfactual transframes from factual preconditions and factual transframes. However, when the time comes to make a prediction of a counterfactual future, given factual preconditions, often the hypothesis spaces will predict a number of potential outcomes with uncertainty. This uncertainty arises because, assuming that the choice of hypothesis language allows for the expression of the correct hypothesis, not enough factual examples have been seen in order to reduce the space to this correct hypothesis. Mitchell explains that if all of the current possible hypotheses in the hypothesis space predict the same outcome, this is guarenteed to be the correct outcome, again assuming that the correct hypothesis is expressible in the hypothesis language. However, often it is the case that the hypotheses make different predictions. If hypotheses are assumed to be equally likely, then they can be counted and a probabilistic ratio can be computed. I do not make the assumption of equal likelihood of hypotheses, but some assumption must be made in order for a decision to be made as to the counterfactual outcome.

The important point is that when a decision is made, counterfactual knowledge is created that depends on a specific subset of the hypotheses in the hypothesis space remaining true. In the implementation, callbacks on the hypothesis-version-space learning algorithm are used to propogate the removal of a hypothesis to the counterfactual knowledge during learning and refining the hypothesis space due to new factual knowledge. This is only part of the response to what will be later discussed as a type of *expectation failure*, which creates a memory of the factual plan failure event in addition to the refinement of the hypothesis space.

12.7 counterfactual transframes

When a counterfactual transframe is decidedly created based on specific subsets of hypotheses in different hypothesis spaces,

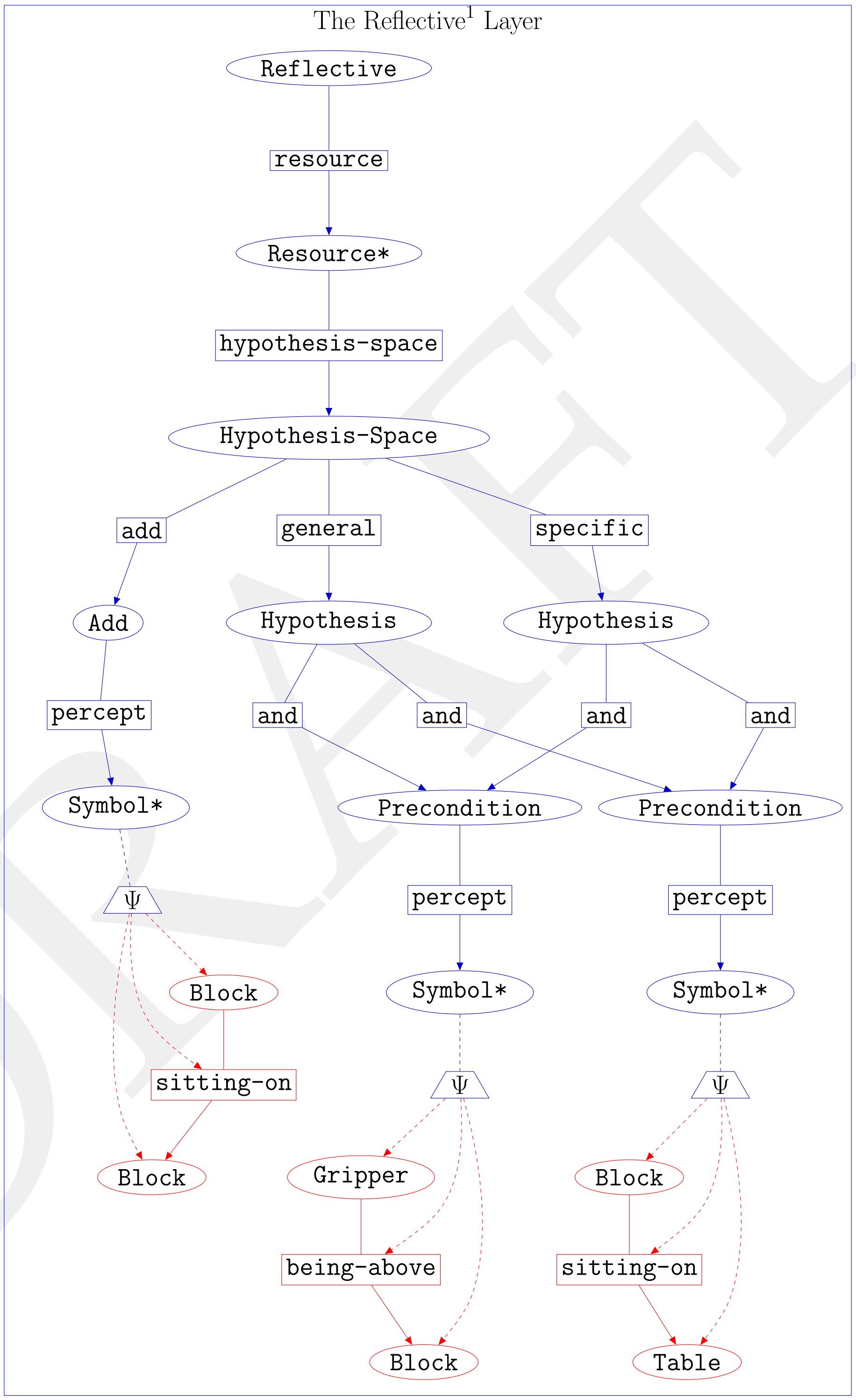


Figure 26: An example of a resource activation hypothesis space that can be used to predict the transframe addition of the percept of a block being on top of a block.

these are kept track of as *hypothesis dependencies* for each counterfactual transframe. Further, counterfactual transframes depend on the future activation of a given set of resources. These *activation dependencies* are also part of a counterfactual transframe. Figure 27 shows an example of a counterfactual transframe with a hypothesis dependency as well as an activation dependency.

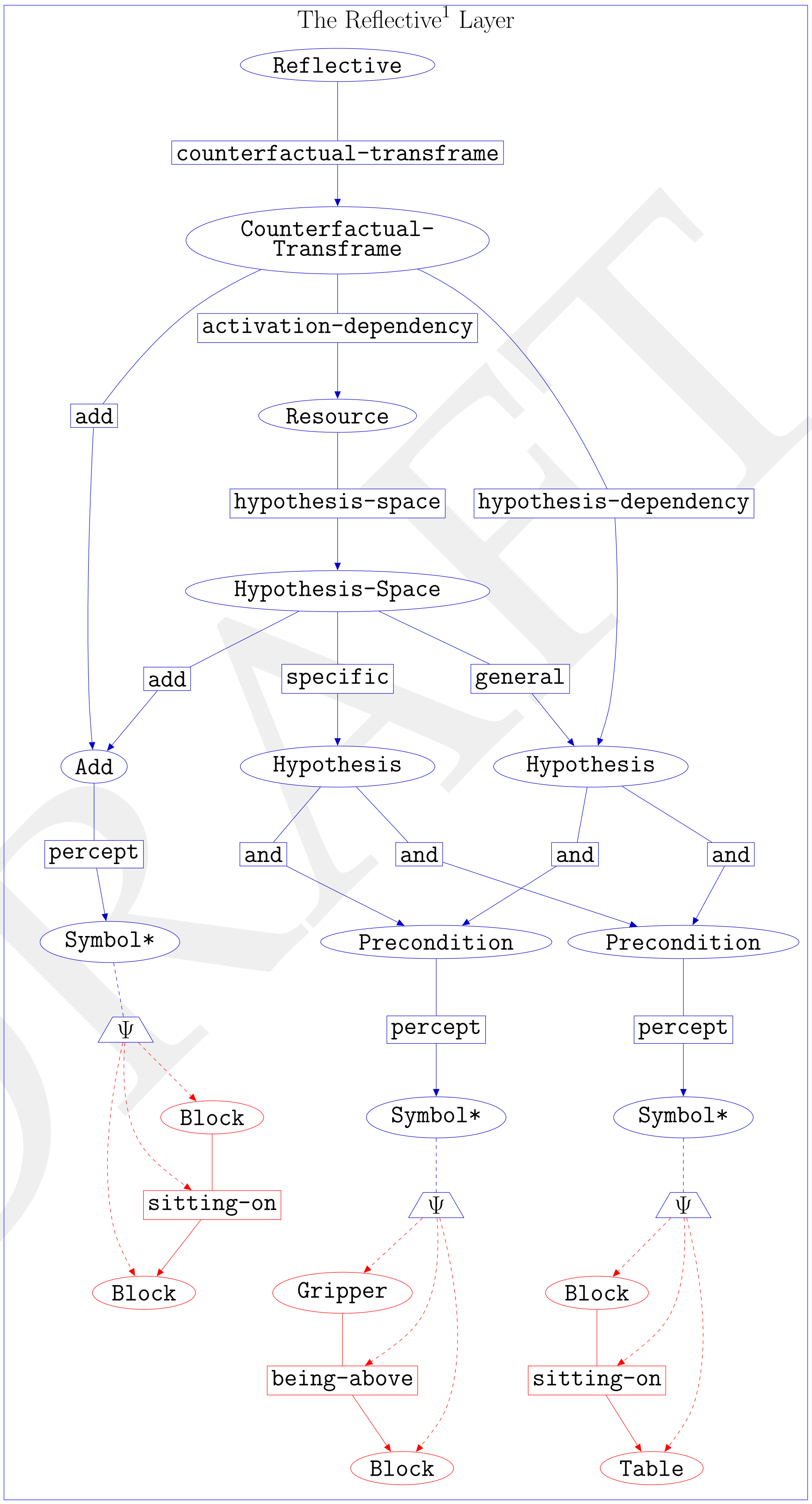


Figure 27: An example of a counterfactual transframe with a hypothesis dependency as well as an activation dependency.

13 THE PLANNING MACHINE

13.1 PLAN OPERATORS

In the implementation, plans are composed of four basic types of plan operators.

1. The *resource activation operator* references a counterfactual transframe, which includes specific resource activation dependencies as well as specific hypothesis dependencies.
2. The *resource suppression operator* references a resource that should be suppressed when this operator is executed.
3. The *sequential program operator* temporally orders two other plan operators.
4. The *parallel program operator* arranges two other plan operators to be executed simultaneously.

Figure 28 shows an example of a plan with four composable plan operators, while Figure 29 shows a more detailed example of a two-step plan.

13.2 A PLANNING MACHINE

The first-order reflective thinking layer makes plans in order to make plans that will accomplish symbolic physical goals when executed. Figure 30 shows an example of a planning machine. In the model, a planning machine is a simple computational machine with different registers that creates and manipulates plan objects. The operations of a first-order planning machine can be seen as symbolic resources in the second-order reflective thinking layer. In general, planning operations can be any kind of dynamic activity of the first-order reflective thinking layer. Because the model is not yet advanced enough to abstract objects and their subjective perspectives, this planning machine the resources in the second-order reflective layer resemble the simple symbolic actions of a RISC processor, which are like the symbolic bytecodes of a virtual machine. The simplest symbolic resources in the second-order reflective layer are the following list of actions that simply move plans between the different registers of the planning machine:

- `focus-on-register-a*`: Copy the plan in `register-a` to the `focus` register.
- `focus-on-register-b*`: Copy the plan in `register-b` to the `focus` register.

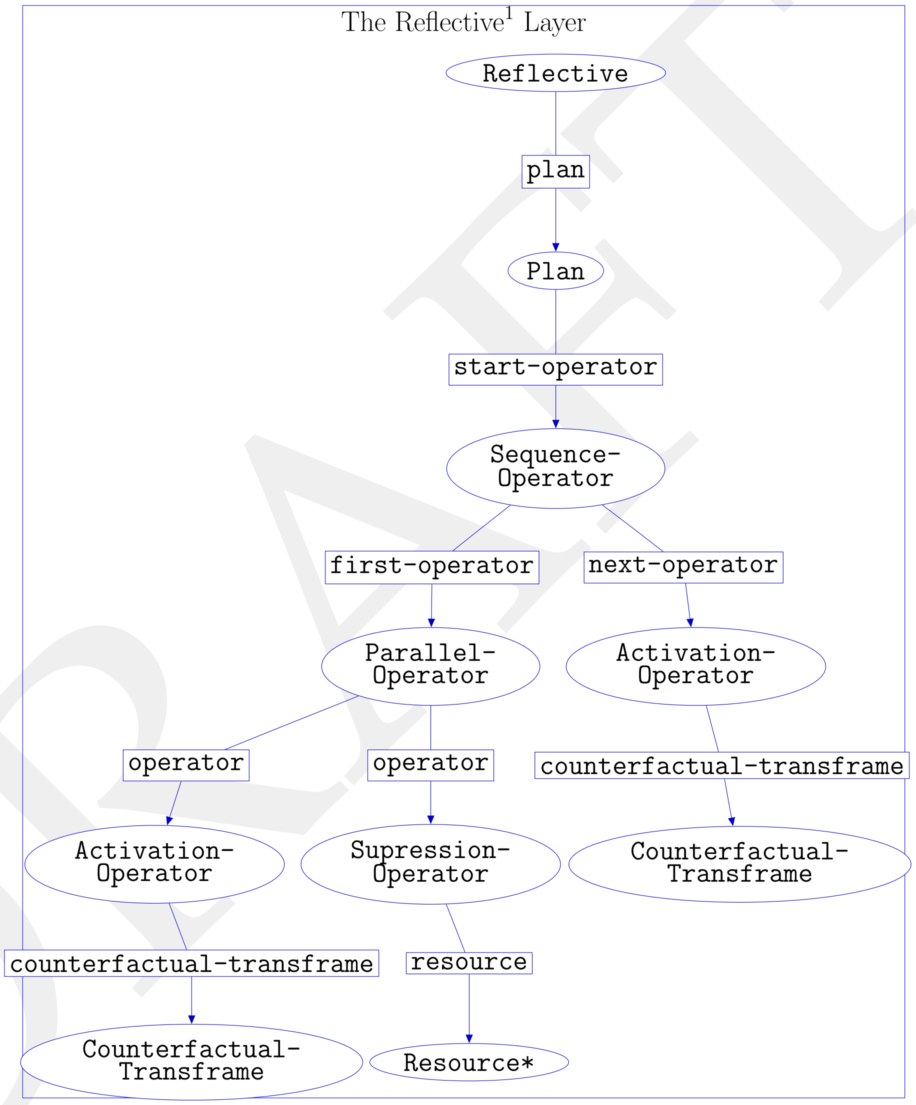


Figure 28: An example of a plan with four basic plan operators: (1) the resource activation operator, (2) the resource suppression operator, (3) the sequential program operator, and (4) the parallel program operator.

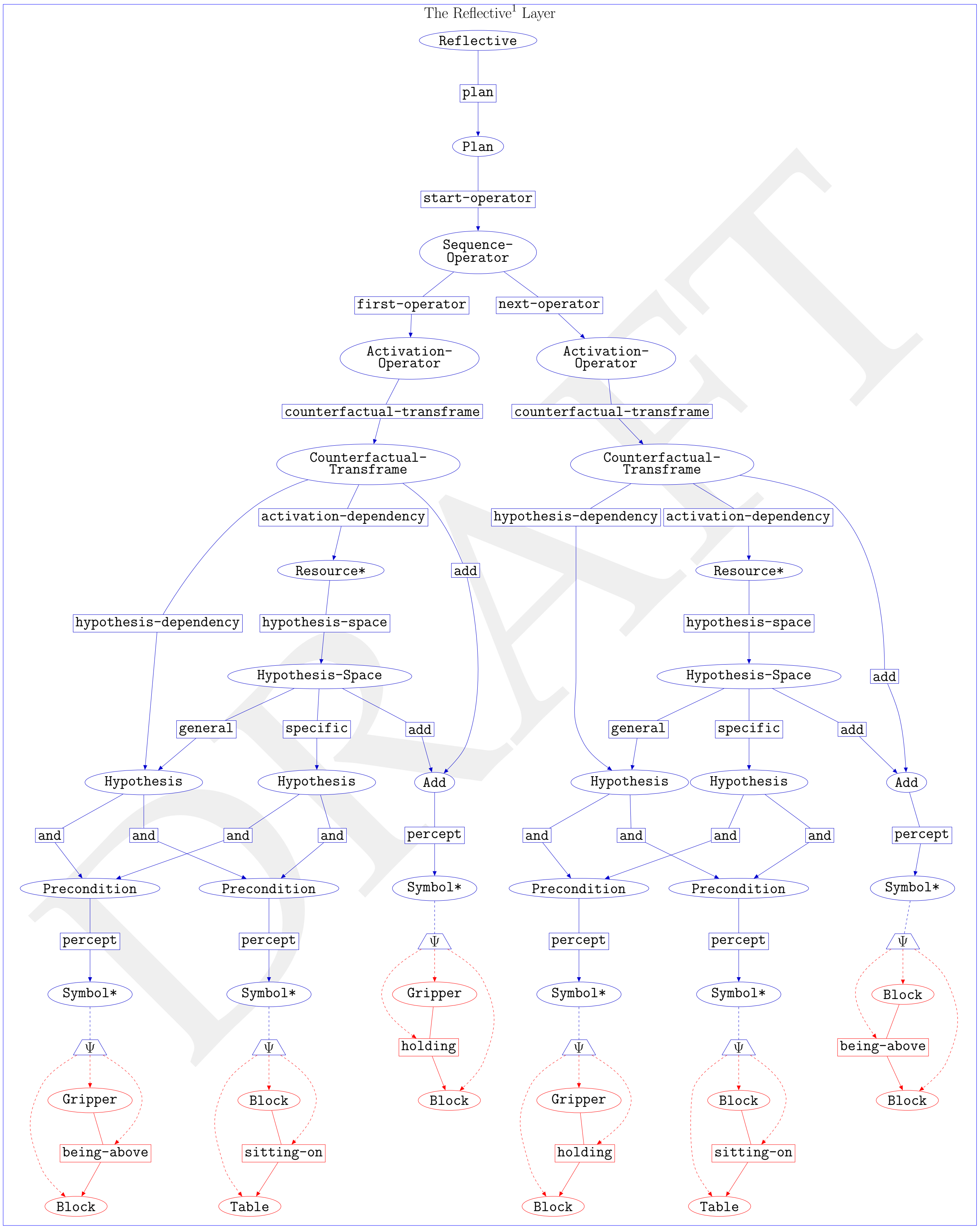


Figure 29: An example of a two-step plan.

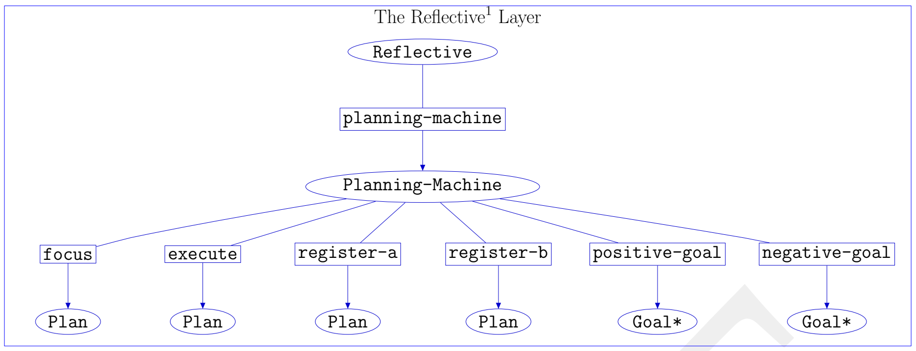


Figure 30: An example of a planning machine.

- `focus-on-execution*`: Copy the plan in the `execute` register to the `focus` register.
- `copy-focus-to-register-a*`: Copy the plan in the `focus` register to register `a`.
- `copy-focus-to-register-b*`: Copy the plan in the `focus` register to register `b`.
- `execute-plan-in-focus*`: Copy the plan in the `focus` register to the `execute` register.
- `execute-plan-in-focus*`: Copy the plan in the `focus` register to the `execute` register.
- `stop-execution*`: Remove any plan from the `execute` register.

The planning machine's execution register is a special register that holds the plan that is currently executing. As the planning machine is like a virtual machine, plans are like computer programs for this virtual machine. The planning machine has a number of activities that create plans that accomplish or avoid the current goals in the `positive-goal` and `negative-goal` slots of the planning machine. These activities are seen as symbolic resources that can be used to accomplish planning goals by the second-order reflective thinking layer. Some of these plan creation and manipulation resources in the second-order reflective thinking layer are as follows:

- `create-empty-plan*`: Create new plan with no operators and place in the `focus` register.
- `extend-plan-greedy*`: Create new plan in the `focus` register by adding a resource activation operator to the end of a duplicate of the plan in `register-a`. The new activation operator is chosen using a greedy technique that compares the options with a heuristic.
- `sequence-plans*`: Create new plan in the `focus` register with a sequential program operator that orders a duplicate of the plan in `register-a` as the `first-operator` and a duplicate of the plan in `register-b` as the `next-operator`.

- `parallelize-plans*`: Create new plan in the `focus` register with a parallel program operator that contains duplicates of the plans in `register-a` and `register-b`.

- `reverse-sequence*`: Create new plan in the `focus` register with the reverse of a sequence operator in the plan in `register-a`. Sussman (1973) mentions a related planning bug where preconditions of one action clobbers the preconditions of another, where the solution is sometimes simply reversing the actions in the plan.

- `parallelize-sequence*`: Create new plan in the `focus` register replacing a parallel program operator with a sequence operator in the plan in `register-a`.

These first-order reflective planning operators can be used by the second-order reflective layer in order to make second-order plans that control the first-order planning machine.

13.3 symbolizing goals

Positive and negative first-order planning machine goals are symbolized from the reflective^0 layer of physical activity. Note the bottom-up type of goal-oriented control in the model. Also, because reflective thinking layers are thought of as running concurrently in the model, there is no necessary priority for devoting thinking resources to any specific layer of thinking as more important than another. The implementation includes built-in goals that have already been symbolized as positive goals that are included as part of the first-order reflective planning machine.

13.4 second-order resources

The second-order reflective layer is the lowest layer that is able to symbolize references to the dynamic activities of reflective thinking. Although not included in the implementation, here is a list of examples of symbolic resources that could exist in the reflective thinking layers of order two or greater:

- `complete-execution*`: Copy the plan in the `focus` register to the `execute` register and wait for the plan to complete execution, resulting in either failures or success.

- `refine-symbolic-perception*`: Create a new symbol that divides the currently perceived symbols into smaller subgraphs of dynamic activity that may be helpful for accomplishing or avoiding the positive and negative goals currently in the planning machine.

- `refine-symbolic-resource*`: Create a new symbol that refers to a subgraph of the current symbolic resource references.

- `create-type-perception*`: Create a new symbol that refers to the logical "or" or disjoint perception of two or more symbolic perceptual references.
- `create-analogical-plan*`: Create a new plan in the `focus` register that satisfies the Winston (1970) difference-of-differences relationship between the plans in `register-a` and `register-b` when applied to `register-c`. This algorithm has been implemented by Panupong Pasupat and is included in the "analogy" package of the implementation, but is not included in my proof of concept demonstrative example.

13.5 CRITICAL FAILURE HEURISTICS

Sussman (1973) introduced the idea of a critical planning heuristic, which is based upon a logical search and pattern recognition algorithm. Building on this work, Singh & Minsky (2003, 2005) describe a variety of critical first-order and second-order reflective resources, many of which assume a logical search over an object oriented social model. I have taken painstaking efforts to not introduce any logical problem solving into my algorithm, so that each basic step in my planning machine that compose more complex planning actions takes a predictable constant time complexity, $O(1)$. This allows all critical and logical search algorithms in my model to be implemented in a way that is reflected upon and optimized by the critical learning algorithm itself. Minsky (2006) describes a "critic-selector" algorithm as follows:

> Each Critic ... can recognize a certain species of "Problem Type." When a Critic sees enough evidence that you now are facing its type of problem, then that Critic will activate what we shall call a "Selector," which tries to start up a set of resources that it has learned is likely to act as a Way to Think that may help in this situation.

I use critics in my model to predict different types of reflective plan execution failures.

13.6 PLANNING FAILURES

Planning may fail in a number of different ways in a reflective layered system. Note that while planning may fail, the reflective^0 physical layer of activity does not include the concept of failure because this layer contains neither symbolic perceptions, actions, nor goals. The physical layer is the given, goals are symbolized in the first-order reflective layer, so this layer of activity is the first layer that may fail to accomplish its own created goals.

The most common type of failure that is discussed in the literature is an expectation failure. This is the type of failure that I demonstrate in the implementation. Cox (2007) discusses a number of different self-reflective failures in story understanding that are based on a logical search algorithm. My model is more basic than most approaches to implementing artificial reflective thinking because it carefully avoids any logical resolution search algorithm that operates behind the scenes. Although, this model does not have a predefined object-oriented (or subject-oriented) view of problem solving, there are still a variety of interesting types of plan failure that occur at this level of reflective debugging. I have discussed two types of first-order plan execution failures, here listed as critical failures followed by a selective response:

- `expectation-failure` $\longrightarrow$ `refine-resource-transframe-hypotheses`
- `activation-conflict-failure` $\longrightarrow$ `create-parallel-suppression-plan`
- `failure-to-distinguish-predictions` $\longrightarrow$ `refine-symbolic-perceptions-and-resources`
- `exhausted-hypothesis-space` $\longrightarrow$ `generalize-hypothesis-language`
- `activation-failure` $\longrightarrow$ `create-new-symbolic-resources`

13.7 factual plan failure knowledge

When the planning machine executes a plan, the plan may encounter a bug, a failure. Failed plans, failed counterfactual knowledge, and their untrue hypothesis dependencies are not immediately discarded in a reflective learning algorithm as is the case with non-reflective learning algorithms. These failure events are factual knowledge about the actual use of counterfactual knowledge. Factual failure knowledge is used to learn hypothetical knowledge about what plan structures in the context of current physical perceptions are prone to specific types of failure. Given a factual plan failure event, different symbolic types of failure can be predicted from given plan structures in the context of the layers below the planning machine. Figure 31 shows an example of a one-step plan with a factual expectation failure structure.

13.8 prediction of failure based on plan structure

The second-order reflective thinking layer creates symbolic references to both the first-order reflective thinking layer as well as the physical layer of activities. Resources in the second-order reflective thinking layer control the activities in layers below, including the planning machine's manipulation of plan structures. Second-order hypothesis spaces are learned in order to predict the transitional effects of first-order reflective planning machine activities. The second-order hypothesis spaces contain hypothetical knowledge about the actual factual use of counterfactual

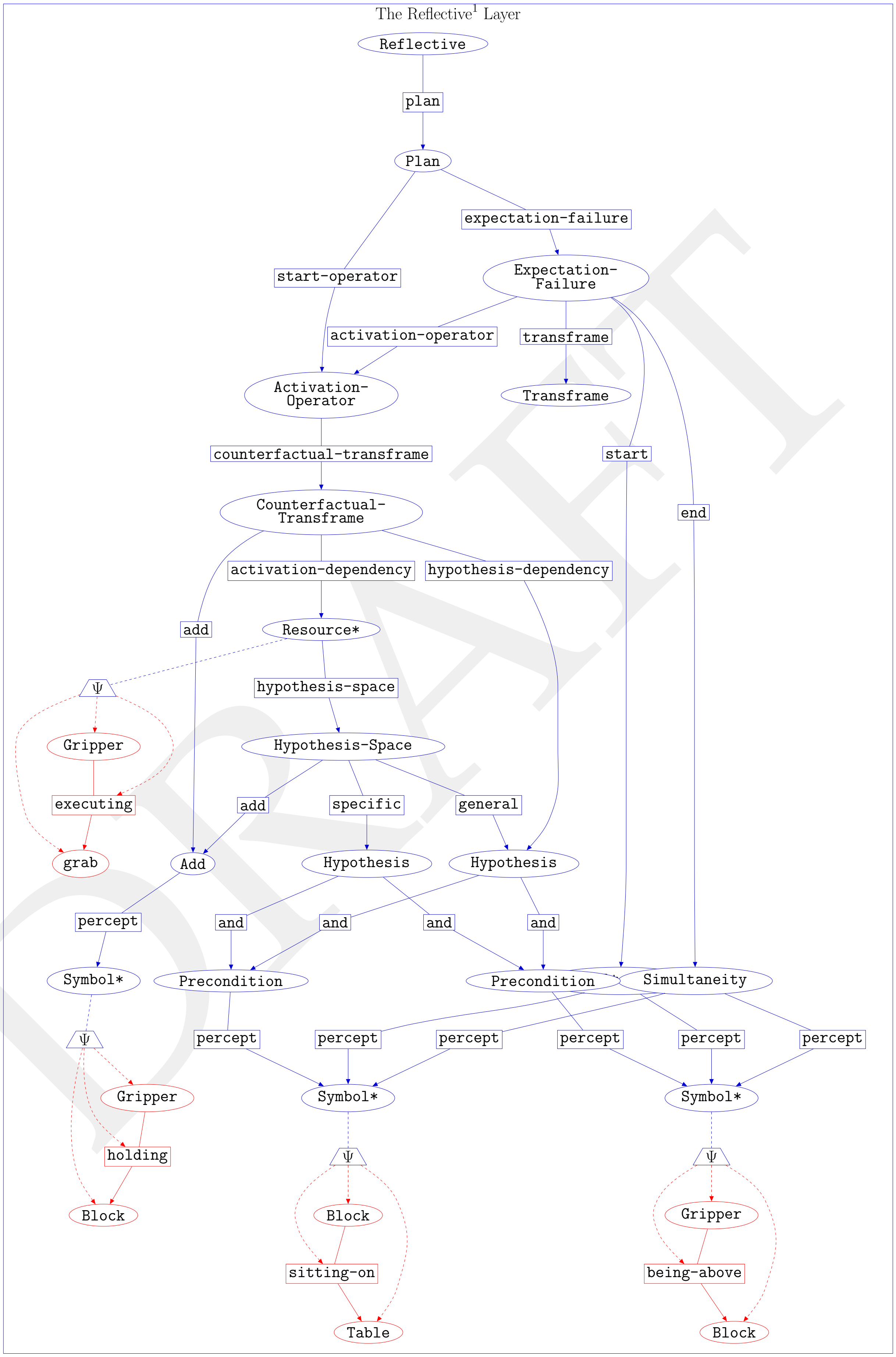


Figure 31: An example of a one-step plan with a factual expectation failure, where the gripper is expected to pick up the block, but the factual transframe shows that nothing actually happens in this case of execution. This could be due, for ex-

knowledge. While goals in the second-order reflective thinking layer can be specifically about physical activities, the power of second-order reflective goals comes from considering goals that are abstracted from this physical layer of activity. For example, a second-order reflective thinking goal could be to control the planning machine in order to work against the first-order goals. The first-order planning machine activities can be thought of as plans being executed by the second-order planning machine. Figure 32 shows an example of planning machine symbolic perceptions.

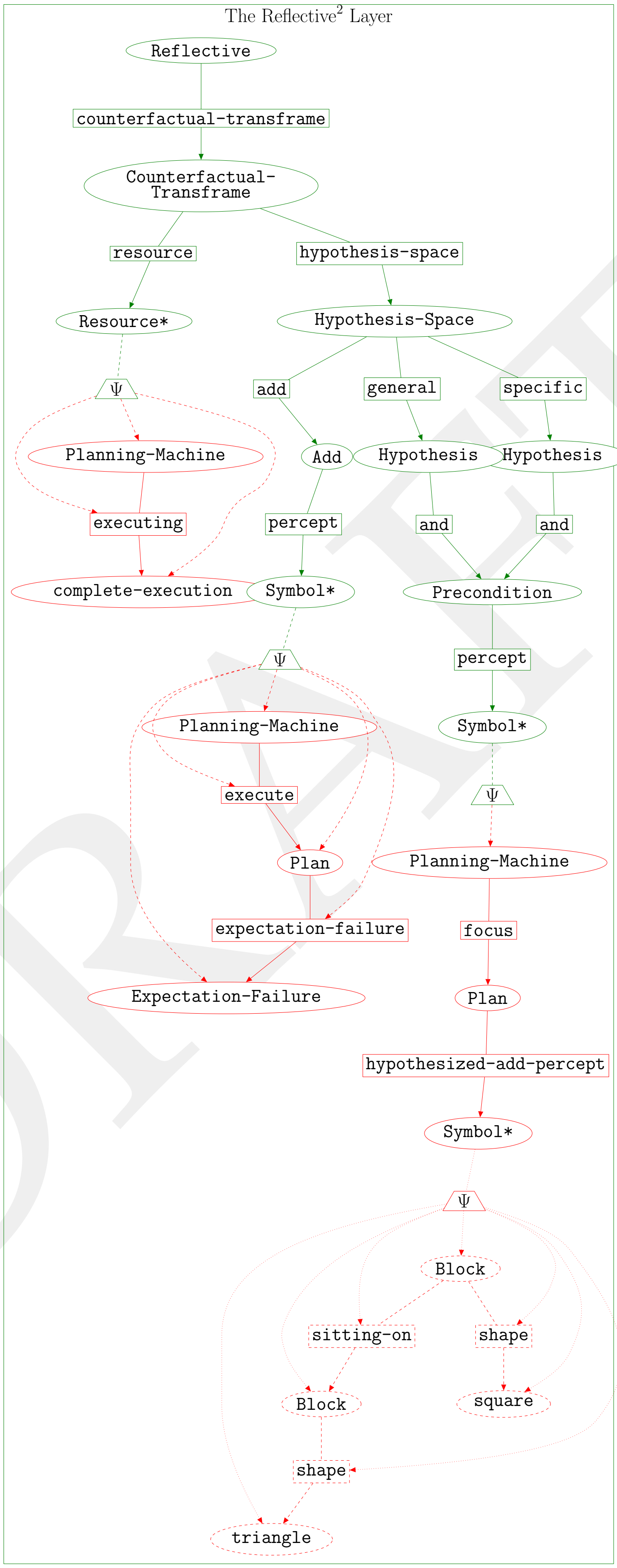


Figure 32: An example of a second-order counterfactual transframe predicting a plan execution failure. In this example, a plan that hypothesizes the creation of a stack of blocks with a square block stacked on top of a triangle block is predicted

Part III

THE IMPLEMENTATION

14 INTRODUCING COMPUTATION

14.1 DIGITAL ABSTRACTION

Computer simulations are based on the *digital abstraction*, an assumption that provides the most basic symbols of a computational model, “1” and “0”. I will refer to the capacity to arrange symbols in Space as *computer memory*. I will refer to a specific arrangement of symbols in computer memory as *data*. While computer memory is often thought of as arising from physical objects, such as transistors, the specific physical derivation of these symbols is not important, given the modelling assumption of digital abstraction. For example, computer memory can be written down on paper and manipulated by hand, once the idea is understood, it is a tool for thinking, separately from any physical implementation. This assumption is necessary when using computers in modelling work. Computer memory is a static arrangement of symbols in Space.

14.2 THE COMBINATIONAL DEVICE

The digital abstraction is not all that is assumed when using a computer to simulate a model. The digital abstraction provides the static symbols in Space, so let us now focus on the dynamic activities in Duration. In simulating a model with a computer, there is a predetermined discrete transition from the past to the future. In the past, there is an arrangement of symbols in Space and there is a predetermined activity in Duration that causes these symbols in Space to be arranged in a different specific configuration in the future. This discrete activity is designed to be exactly the same for every transition from the past to the future. This logical predetermined transition is referred to as *the combinational device*. The combinational device maps any specific combination of symbols Spatially arranged in the past to a specific Spatial arrangement in the future.

14.3 DIGITAL REFLECTION

Because the combinational device is always understood to be the exact same activity, there is not a non-tautological reason to symbolically reference this activity in the computer memory. Because this transitional activity is always understood to be exactly the same, the symbolic cause that refers to the activity during the transition from the past to the future would be predefined and thus always the same, making no distinctions. I will refer to the ability to symbolize the ongoing dynamic activity of the digital transition from the past to the future as *digital reflection*.

14.4 the programmer

Symbols have grounding only in reference to the ongoing activities in Duration. So, then, the question becomes: to what dynamic activities of Duration do digital symbols refer? The symbols in computer memory are programmed by a programmer that uses these symbols as references to the activities in Duration that are actively ongoing and currently being experienced by the programmer. Computer memory is a static tool of thought for the programmer, who provides the grounded reference to the dynamic for the 1's and 0's in the computer memory. Again, a computer does not create symbols that reference the dynamic. Given symbols that have meaning to the programmer, these symbols are used to create models that are simulated according to the dynamic activities of the combinational device. The model that I have programmed in this way simulates the reflective creation of simulated symbols, or symbols*. These symbols* can then be dynamically displayed on a computer monitor, symbolically perceived, and given meaning outside of the computational process by the programmer or other observer.

15 COMPUTATIONAL ACTIVITIES IN DURATION

My implementation is based on a virtual machine that allows parallel and concurrent scheduling of Lisp-like programs. I refer to this virtual machine as *SALS*, the substrate for accountable layered systems, the focus of this dissertation. The purpose of SALS is to provide a simulation of the actual dynamic activities in Duration, the activities in Duration that actually exist.

In this dissertation, I am focusing on how SALS demonstrates reflective learning in two distinct causal classes of knowledge from a single physical expectation failure. I must begin my description with the dynamic activities in Duration. Thus, first, I will describe how SALS simulates the activities in Duration. I will explain the symbolization of these activities in the context of a block building physical simulation. I will describe how causal models are learned. I will give examples of how causal models are put together into plan objects that are simple serial and parallel SALS programs. I will explain how a plan may begin executing and fail. Finally, I will describe how the expectation failure response updates causal models in both reflective layers. This, in total, shows how learning how to plan is simulated by SALS.

15.1 THE FUNK VIRTUAL OPERATING SYSTEM

The virtual concurrent machine and operating system underlying SALS is called *Funk* for two reasons: (1) the "slap" style of bass guitar playing (Graham 1969, Johnson 1984), which emphasizes the fundamentals of the dynamic actual ongoing, and (2) the focus on reflective tracing of concurrent procedural "funktions" as opposed to purely declarative or logical approaches, which have no means of optimizing the underlying search problem. In this dissertation, I focus on the implementation of second-order reflective thinking within this virtual operating system. In order to come to this point, a very general underlying substrate has been programmed, which will largely not be discussed. To briefly give an idea of the complexity of the implementation, the following table gives quantification of the size of the SALS codebase:

Lines of C code:	141,138
Lines of Funk code:	50,346
Total lines of code:	192,868
Total characters of code:	10,117,671

The project is small compared to other virtual operating systems and programming languages, but represents a non-trivial effort toward a fully reflective parallel and concurrent virtual operating system for implementing large-scale practical applications, such as word processors, web browsers, as well as scientific data processing tools.

15.2 THE GLOBAL ENVIRONMENT

In SALS, the simulation state graph, S, becomes the *global environment*. The global environment is SALS' static graph representation of the dynamic activities in Duration. The previous part described the simulation state as the mathematical representation for the mind. I previously defined this simulation state graph to represent all subgraphs that actually exist in Equation 9.9, reproduced here:

$$\text{exists}(G) \longleftrightarrow (G \subseteq S)$$

Environments are organized into hierarchies that have two parts: (1) a frame containing slots containing variable definitions, and (2) a reference to a parent environment. The global environment is a special environment because it is the only environment that has no parent environment. The entire SALS computational implementation is contained within the static global environment or its directed references.

15.3 THE GLOBAL SCHEDULER

SALS is able to take advantage of multi-core and multi-processor hardware, which makes it a concurrent processing system. SALS explicitly represents these hardware processors in an object called the *global scheduler*. The global scheduler is contained in the global environment frame under the slot name `global-scheduler`. The global scheduler object contains references to a number of processors that reflect the number of actual hardware processors in the current system. For example, the implementation has been successfully tested on 1 through 32 processors, although most of the demonstration simulations were computed using a machine with two processors, each with 4 cores, which appears as 8 concurrent processors within SALS.

15.4 FIBERS SIMULATE ACTIVITIES

SALS introduces an object that represents a virtual machine that can run user programs, the *fiber*. Fibers are the fundamental element for simulating parallel user programs in SALS. Each concurrent processor contains references to a collection of fibers that it is responsible for simulating. Fibers execute bytecode programs. In each concurrent processor's execution cycle, it runs a

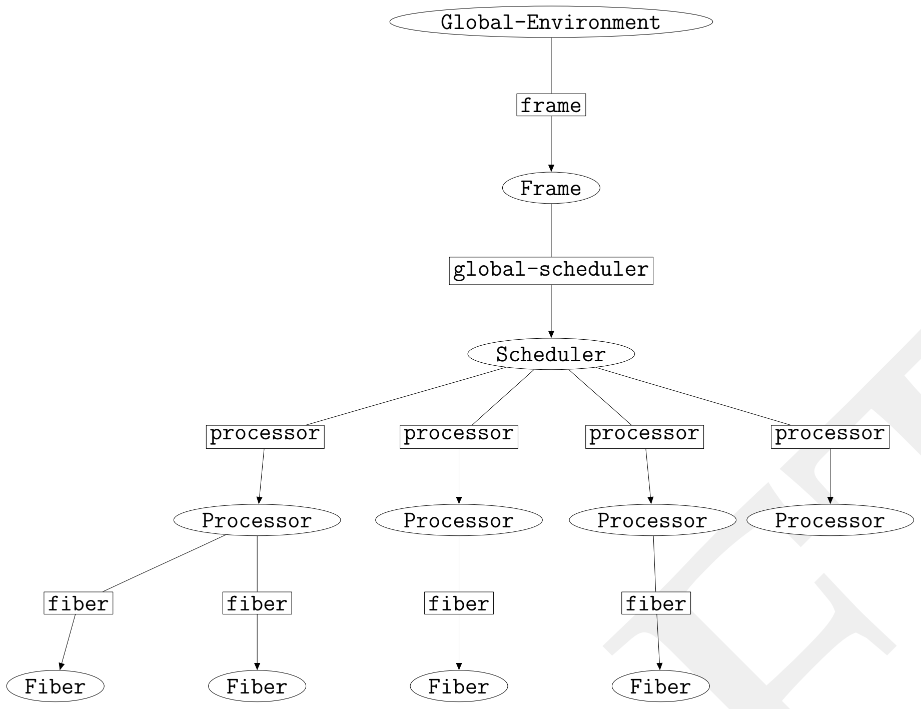


Figure 33: An example of a global environment with its frame object and a definition of the global scheduler object. The global scheduler contains references to concurrent processors, which organize the parallel fibers, virtual machines that execute user programs.

given finite number of bytecodes for each of its fibers, simulating parallel execution.

SALS begins the simulation with only one fiber that serves a predefined "boot-up" function that starts all of the other concurrent fibers that are necessary for simulating the mind. Fibers are the discrete elements that I have used as a model of the concurrent, actually inseparable, activities in Duration.

My simulation of mind uses hundreds of thousands of fibers in order to demonstrate two layers of reflective learning. Fibers are often very short-lived processes; for example, not execution, but simply compiling a single Lisp-like expression in SALS can lead to the creation and destruction of hundreds of fibers of activity. Fibers are generally useful in programs that could use an extra perspective on a problem. Figure 33 shows an example global environment, containing the global scheduler, concurrent processors, and parallel executing user-programmable fiber objects.

15.5 STEPPING THE SIMULATION STATE

For the purposes of description and evaluation of the complexity of my implementation of the reflective simulation, I assume that each concurrent processor executes at the same speed and steps the simulation forward at the same time. Given a simulation state graph, S, all of the concurrent processors step the simulation forward to the next simulation state graph. Equations 15.1 and 15.2 define the initial simulation state and recursive definition of the step function that computationally creates the next simulation state graph.

$$S[0] \equiv \textit{Initial simulation state graph} \tag{15.1}$$
$$S[n+1] = \text{step}(S[n]) \tag{15.2}$$

Note that until this point, I have not previously introduced a computational step function for the simulation state. I have previously defined the temporal transitions that compose n layers of distinct temporal orderings, one for each reflective layer in the model, but these layers of temporal transition representations are within the static state of the mind. This computational step function allows all of the previously described static structures to change to new static representations. The computational step function does not exist within the mind, and the mind does not have any access to symbolizing these different simulation state graphs that are now introduced in the computational implementation.

Causally traced procedural reflection is the key component that allows SALS to monitor changes to its own simulation state graph representation. Before discussing how procedural reflection must be embedded into the definition of SALS' fundamental memory representation, I must first briefly describe how fibers are programmed in order to simulate simple non-reflective programs.

15.6 PROGRAMMING LANGUAGE

SALS includes a Lisp-like programming language, which is programmed by typing statements that are called *expressions*. If expression 15.3 were typed into SALS, the symbol "green" would be printed to the user's terminal screen.

```
[print 'green]                                              (15.3)
```

The `print` command is a useful debugging tool that can report status messages to the programmer as the fiber reaches a specific point in the program.

15.7 SEQUENTIAL AND PARALLEL PROGRAMS

SALS includes expressions for describing serial and parallel programs. For example, the expression

```
[prog [print 1]
      [print 2]
      [print 3]]
```

results in the output trace

```
1
2
3.
```

The command prog is a way for expressing a sequence of commands to be executed in serial order. SALS also includes the parog command for executing a list of commands in parallel, waiting for them all to complete, and then continuing.

```
[parog [print 1]
       [print 2]
       [print 3]]
```

When parog is used, it is unclear what command will complete first because they are all running concurrently, in parallel, starting at slightly different times. Here is an example output trace from the parog expression:

```
3
1
2.
```

The parog expression is one way to easily start a number of parallel fibers to simultaneously execute a number of different tasks and wait for these tasks to complete.

15.8 fibers create objects

SALS includes an object type system. Every expression in SALS, besides nil, has an object type. Objects in SALS are in most cases based on a frame knowledge representation. Objects have three main classes of functionality: get, set, and have. The uses of these types of functionality are as follows:

- `[get <object> <slot-name>]`

  The object-oriented get command retrieves the value from the named slot of the object's frame.

- `[set <object> <slot-name> <new-value>]`

  The object-oriented set command mutates the value at the named slot of the object's frame to be the given new value.

- `[have <object> <slot-name> <arguments>]`

  The object-oriented have command performs other types of activities that are not simple frame slot accessors and mutators. The have commands sometimes involve complex mutations or other side-effects.

15.9 creating parallel fibers

The apply operator is the normal way for a SALS program to evaluate a given function with arguments:

```
[apply <function> <arguments>]
```

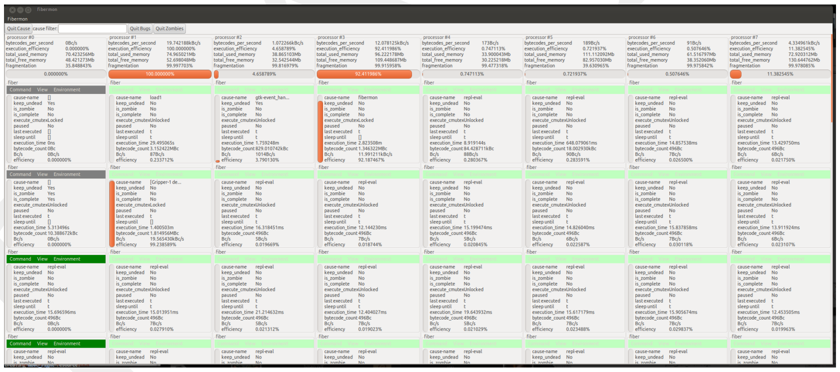
The fundamental operator of SALS's parallel programming language is the `fiber` operator:

```
[fiber <function> <arguments>]
```

The `fiber` operator does not evaluate the given function with arguments. The `fiber` operator starts a new parallel fiber that will evaluate the given function and arguments; the new fiber object is returned. The intermediate state or final result can be monitored through the fiber object.

15.10 MONITORING SIMULATED ACTIVITIES

SALS includes a tool named *FiberMon* that is written in the SALS programming language. FiberMon helps the programmer to introspect on all fibers currently in the simulation scheduler and is shown in Figure 34. Fibers may be easily removed or added to the scheduler, which enables efficient scheduler optimizations to be implemented at the highest levels of the language. If any bugs arise in any parallel fiber, that fiber shows up in red in the monitoring application, so that it may be inspected and debugged by hand.



```
processor #0
bytecodes_per_second   0Bc/s
execution_efficiency   0.000000%
total_used_memory      70.423256Mb
total_free_memory      48.421273Mb
fragmentation          35.848843%
0.000000%
fiber
cause-name        []
keep_undead       Yes
is_zombie         No
is_complete       No
execute_cmutexLocked
paused            No
last executed     []
sleep until       []
execution_time    0ns
bytecode_count    0Bc
Bc/s              0Bc/s
efficiency        0.000000%
fiber
cause-name        []
keep_undead       Yes
is_zombie         Yes
is_complete       Yes
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       []
execution_time    5.313496s
bytecode_count    10.388672kBc
Bc/s              0Bc/s
efficiency        0.000000%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    15.696596ms
bytecode_count    496Bc
Bc/s              0Bc/s
efficiency        0.000000%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
```

```
processor #1
bytecodes_per_second   19.742188kBc/s
execution_efficiency   100.000000%
total_used_memory      74.965021Mb
total_free_memory      52.698048Mb
fragmentation          99.997703%
100.000000%
fiber
cause-name        load1
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    29.495065s
bytecode_count    3.152422MBc
Bc/s              87Bc/s
efficiency        0.233712%
fiber
cause-name        [Gripper-1 de...
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexLocked
paused            No
last executed     t
sleep until       []
execution_time    1.400503m
bytecode_count    1.814956MBc
Bc/s              19.565430kBc/s
efficiency        99.238589%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    15.013951ms
bytecode_count    496Bc
Bc/s              7Bc/s
efficiency        0.027910%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
```

```
processor #2
bytecodes_per_second   1.072266kBc/s
execution_efficiency   4.658789%
total_used_memory      38.865103Mb
total_free_memory      32.542544Mb
fragmentation          99.816979%
4.658789%
fiber
cause-name        gtk-event_han...
keep_undead       Yes
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    1.759248m
bytecode_count    829.010742kBc
Bc/s              1014Bc/s
efficiency        3.790130%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    16.318451ms
bytecode_count    496Bc
Bc/s              5Bc/s
efficiency        0.019669%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    21.214632ms
bytecode_count    496Bc
Bc/s              5Bc/s
efficiency        0.021312%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
```

```
processor #3
bytecodes_per_second   12.078125kBc/s
execution_efficiency   92.411986%
total_used_memory      96.222178Mb
total_free_memory      109.448687Mb
fragmentation          99.915958%
92.411986%
fiber
cause-name        fibermon
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexLocked
paused            No
last executed     t
sleep until       []
execution_time    2.823508m
bytecode_count    1.346322MBc
Bc/s              11.991211kBc/s
efficiency        92.187467%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    12.144230ms
bytecode_count    496Bc
Bc/s              7Bc/s
efficiency        0.018744%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    12.404027ms
bytecode_count    496Bc
Bc/s              7Bc/s
efficiency        0.019023%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
```

```
processor #4
bytecodes_per_second   173Bc/s
execution_efficiency   0.747113%
total_used_memory      33.900043Mb
total_free_memory      30.225218Mb
fragmentation          99.477318%
0.747113%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    8.919144s
bytecode_count    84.428711kBc
Bc/s              88Bc/s
efficiency        0.280367%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    15.199474ms
bytecode_count    496Bc
Bc/s              5Bc/s
efficiency        0.020845%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    19.643932ms
bytecode_count    496Bc
Bc/s              5Bc/s
efficiency        0.021029%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
```

```
processor #5
bytecodes_per_second   189Bc/s
execution_efficiency   0.721937%
total_used_memory      111.112092Mb
total_free_memory      82.957030Mb
fragmentation          39.630965%
0.721937%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    648.079061ms
bytecode_count    18.002930kBc
Bc/s              90Bc/s
efficiency        0.283591%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    14.826040ms
bytecode_count    496Bc
Bc/s              6Bc/s
efficiency        0.022587%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    15.617179ms
bytecode_count    496Bc
Bc/s              7Bc/s
efficiency        0.023488%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
```

```
processor #6
bytecodes_per_second   91Bc/s
execution_efficiency   0.507646%
total_used_memory      61.516797Mb
total_free_memory      38.352060Mb
fragmentation          99.975842%
0.507646%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    14.857538ms
bytecode_count    496Bc
Bc/s              7Bc/s
efficiency        0.026500%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    15.837858ms
bytecode_count    496Bc
Bc/s              7Bc/s
efficiency        0.030118%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    15.905674ms
bytecode_count    496Bc
Bc/s              7Bc/s
efficiency        0.029837%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
```

```
processor #7
bytecodes_per_second   4.334961kBc/s
execution_efficiency   11.382545%
total_used_memory      72.920312Mb
total_free_memory      130.644762Mb
fragmentation          99.978085%
11.382545%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    13.429750ms
bytecode_count    496Bc
Bc/s              6Bc/s
efficiency        0.021750%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    13.911924ms
bytecode_count    496Bc
Bc/s              6Bc/s
efficiency        0.023107%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
is_complete       No
execute_cmutexUnlocked
paused            No
last executed     t
sleep until       t
execution_time    12.453505ms
bytecode_count    496Bc
Bc/s              7Bc/s
efficiency        0.019963%
fiber
cause-name        repl-eval
keep_undead       No
is_zombie         No
```

Figure 34: FiberMon application monitoring hundreds of parallel fibers, simulated activities in Duration, executing on eight different physical processor cores.

15.11 PAUSING AND RESUMING ACTIVITIES

SALS includes a `pause` command to pause the current fiber:

```
[pause]
```

The `pause` command causes the executing fiber to remove itself from the scheduler, after removing itself, it yields the rest of the scheduler cycle. A fiber that is paused in this sense uses none of SALS's processor resources because it is completely removed from the scheduler.

If a fiber has paused, it cannot restart itself. In order for a fiber to resume execution, another executing fiber must use the `resume` command with the fiber object as an argument:

```
[resume <fiber>]
```

The `pause` and `resume` commands are extremely efficient for managing large simulations in which most fibers will be inactive at any given time, but they are a little cumbersome, so I have written a lightweight and helpful *fiber trigger* object, which I will explain after explaining one more object that is used to program fiber triggers.

15.12 mutual exclusion

SALS provides a primitive object called a *mutex* for handling the mutually exclusive access to resources. A mutex object has two possible states: locked and unlocked. If the mutex object is unlocked then when a fiber tries to locks the mutex, the mutex switches to the locked state and the fiber continues execution. If the mutex object is already locked when a fiber tries to lock the mutex, the fiber will pause until the mutex is unlocked, at which point the mutex will be locked and the waiting fiber will resume execution. Mutexes are used for protecting shared resources from being used by more than one fiber at a time.

Mutexes are a special type of object that must be supported by the hardware of a concurrent computer. Almost every parallel programming language has a mutex construct that derives from this hardware mutex. So, mutexes are common to parallel programming, but they are notoriously difficult to debug, resulting in bugs affectionately referred to as "race conditions" or "deadlocks". In order to help with debugging these types of problems, mutexes in SALS keep track of which fibers are waiting for the lock or holding the lock. I have found this extra information invaluable in debugging complicated mutex related bugs.

15.13 fiber triggers

The fiber trigger organizes sets of paused fibers so that they can be woken up at the same time. Basically, the fiber trigger is an object that provides a useful abstraction for controlling fiber execution that combines a mutex with the `pause` and `resume` commands.

A fiber can be added to the resume set of a fiber trigger by using the `wait-for-trigger` command as in the following example:

```
[wait-for-trigger <fiber-trigger>]
```

The `wait-for-trigger` command atomically pauses the current fiber and then adds the fiber to resume queue of the fiber trigger object.

15.14 fiber complete and bug-found triggers

Fibers have two primary ways of completing their execution: by (1) successfully reaching the end of their execution task, or by (2) encountering a bug. Each fiber has a special fiber trigger for each of these two possible events: (1) a `complete` trigger, and (2) a `bug-found` trigger. When a parallel fiber is created, further parallel fibers can be created in order to listen for one or both of these triggers. Within the internals of the Funk operating system, an object called a *larva* is created when a C function encounters an error. A larva object is similar to an exception in other programming languages such as C++ or Java. The larva object propagates up the C execution stack until it reaches the `value` register of the Funk virtual operating system fiber object. The Funk virtual operating system never sees the larva object because when the C code detects a larva object in the `value` register of the virtual machine represented by the fiber object, the larva object is immediately converted to a bug object and the fiber is also immediately paused by the C code. When a bug is encountered the `bug-found` trigger is triggered, and a reflective parallel fiber can respond and inspect the paused fiber that contains the bug in its `value` register. Larva and bugs are very versatile frame-based objects, so they can pass an arbitrary amount of information from the depths of the C code internals of the virtual machine to the very abstract reflective thinking capabilities of the SALS first-order reflective learning-to-act and second-order reflective learning-to-plan systems.

16 CAUSALLY REFLECTIVE PROCEDURAL TRACING

The parallel fiber virtual machines create, mutate, and read from memory as they execute sequences of bytecodes. At any given point, the SALS memory system is static. The current execution state of every fiber is represented in the global environment. In order for SALS to be fully reflective on all of the procedural effects of any given fiber, I introduce a technique that I call *causal reflective tracing*. Causal reflective tracing is simply a way of defining variables that are specific to each fiber that can be used to control the low-level memory access, mutation, and creation functions. This allows one fiber within SALS to subscribe to the procedural trace events of another fiber without receiving procedural trace events of its own execution, which would lead to an infinite regress, halting the system. Further, because SALS is inherently a parallel processing system, a given fiber will often start a number of child fibers that handle part of the processing for the parent fiber. When a new fiber is created, the child fiber inherits the causal variable bindings of its parent fiber, enabling the same procedural tracing options for the child as well. So, causal reflective tracing is one of the basic tools for keeping track of which pieces of memory were created, mutated, or read by which other fibers.

Creating procedural trace events for the execution of a given fiber slows the fiber down by a constant factor. This is an important point to consider when evaluating the theoretical time complexity of concurrent procedurally reflective control algorithms.

16.1 SEMANTIC MEMORY FOCUSES REFLECTIVE TRACING

While causal reflective tracing focuses the procedural event tracing to memory interactions of specific fibers, this still results in millions of events to consider every real-time second of execution. In order to further focus the reflective tracing focus to specific objects within the SALS memory system, specific pieces of memory can be marked as *semantic memory*. Semantic memory objects are created, mutated, and accessed in roughly the same way as all of the frame-based objects in the SALS memory system. Semantic memory objects provide event streams that can be subscribed to by a number of different parallel listeners in different fibers. The following code example shows how frame-based objects with default slot values and constructors are used to define a `Planning-Machine` type object:

```
[deframe Planning-Machine [layer
                               [focus          nil]
                               [execute        nil]
                               [register-a     nil]
                               [register-b     nil]
                               [register-c     nil]
                               [positive-goals nil]
                               [negative-goals nil]]
  [new [initial-layer]
    [= layer initial-layer]
    nil]]
```

Once a frame-based object is defined, the "`new`" operator is used in order to create a new instance of the object type as in the following example:

```
[new Planning-Machine ‘Reflective-1]
```

16.2 forgetful event streams

By default, when there are no listeners to the procedural event streams of a semantic frame-based object, no reflective events are created, allowing the use of the object to run at full speed. When a listener subscribes to the procedural use of a specific semantic memory object, events are added to ordered streams for the listening subscribers. In order to conserve memory resources, when multiple parallel listeners are subscribed to a given event stream, only those events that have not already been seen by all of the subscribers are remembered. Once all subscribers have processed an event, all events before this event are forgotten. This type of memory conserving event stream is referred to as a *forgetful event stream*. In this way semantic frames report the addition and removal of slot values to reflective forgetful event stream subscribers.

16.3 semantic knowledge-bases

Because it becomes awkward to subscribe to each an every frame-based object that may be interesting to the reflective focus, semantic frames that are created by specific fibers can be added to collections of semantic frames that are called *semantic knowledge-bases*. Semantic knowledge-bases are good for organizing entire layers or subsets of reflective layers that contain different types of semantic frames. Semantic knowledge-bases allow the same forgetful event stream subscription services as semantic frames with the additional capability of tracing the addition and removal of entire semantic frames to and from the knowledge-base.

16.4 reflectively reconstructed semantic knowledge-bases

Each parallel subscriber to the reflective procedural knowledge-base events receives each event slightly after it occurs. It often becomes necessary to be able to refer to the historical state of the knowledge-base under the reflective focus. However, in order to not slow down the primary procedure, the primary knowledge base must be allowed to change during the time that it takes the reflective process to process each event. In this case, it becomes necessary to maintain accurate reflective reconstructions of knowledge-bases that are accurate at the reflective time of each reflective event that is processed in parallel. For example, in order to implement symbolic perception, each slot value addition to the physical knowledge-base is reflectively traced and the physical knowledge base is reconstructed as a reflective copy that is accurate to the reflective time. This *reflectively reconstructed semantic knowledge-base* is used to perform a check of the immediate frame-based graph surroundings of the change to see if one of the symbolic perceptions has become active or inactive due to the change.

16.5 semantic event interval-tree knowledge-bases

While knowledge-base reconstructions are extremely fast to reference, $O(1)$, they require a duplication of the memory requirements of the focus knowledge base for every different point in time that is required. In order to allow efficient access of the state of knowledge-bases at arbitrary points in the past, *semantic event interval-tree knowledge-bases* are another type of representation that is reflectively maintained without slowing down the procedure under reflective focus. A semantic event knowledge-base stores a type of semantic frame that represents when a given semantic frame has a specific slot value called a *semantic event*. A semantic event is a semantic frame is an interval that spans a time from a beginning time and an ending time, each of which may or may not be specified. Semantic event knowledge-bases are reflectively traced and the knowledge is always stored in two different representations, the basic semantic event frames as well as a balanced interval tree that always represents the current state of the semantic event knowledge-base. The balanced interval tree allows accessing the state of the focus knowledge-base in $O(\log n)$ time, where n is the number of events stored in the semantic event knowledge base. Although the time complexity is not as efficient as the constant, $O(1)$, access time of the reflectively reconstructed semantic knowledge-base, the semantic event interval-tree knowledge base only requires $O(n)$ memory complexity in order to allow access to the structure of the knowledge-base at any point in the past, where n is the number of semantic events.

17 THE MIND MONITOR APPLICATION

In order to control the SALS model of mind, a mind monitoring application called *MindMon* has been implemented. MindMon is based on abstract physical world and reflective thinking components that allow reflective thinking layers to be interchanged with different reflective^0 physical layers. In this dissertation I focus on a block building domain in order to demonstrate second-order reflective learning to plan, but other more complex physical domains have also been developed within SALS such as the Isis-World physical simulator described by Smith & Morgan (2010), where MindMon allows attaching multiple reflective thinking models to a shared reflective^0 physical layer, a type of simulation that assumes a more complex subjective philosophy of mind that is not assumed in the non-subjective model described in Part I of this thesis. Figure 35 shows the implemented MindMon application.

17.1 THE PHYSICAL KNOWLEDGE-BASE

The block building domain is a simulation based on two-dimensional ridid-body physical laws, including floating point numerical representations for object positions, velocities and accellerations. These numerical representations and the processes that manipulate them are part of the reflective^0 physical layer of the model, but in order to focus the efficiency of the procedural reflection, a much simpler relational graph representation has been specifically represented in semantic frame-based objects in a semantic knowledge-base, referred to as the *physical knowledge-base*. The physical knowledge-base includes objects with shapes, colors, and multiple prepositional spatial relationships. Figure 36 shows the implemented physical knowledge.

17.2 SEMANTIC EVENT KNOWLEDGE-BASE

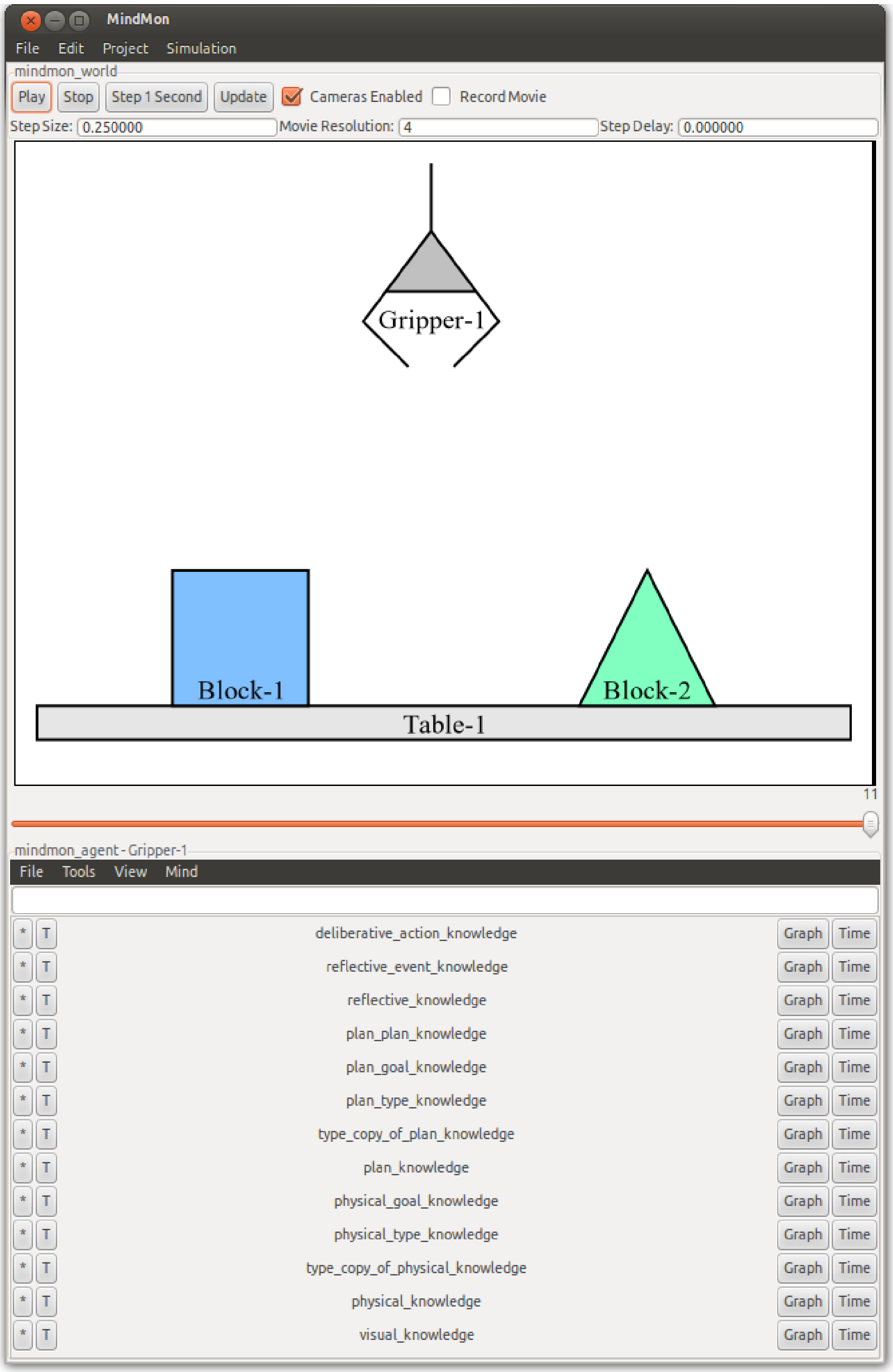
Figure 37 shows the implemented semantic event knowledge base.

17.3 FIRST-ORDER RESOURCE ACTIVATOR

Figure 38 shows the implemented first-order resource activator.

17.4 FIRST-ORDER PLANNING MACHINE KNOWLEDGE

?? shows the implemented first-order planning machine knowledge.



Cameras Enabled: ✓
Record Movie: ☐
Step Size: 0.250000
Movie Resolution: 4
Step Delay: 0.000000

deliberative_action_knowledge
reflective_event_knowledge
reflective_knowledge
plan_plan_knowledge
plan_goal_knowledge
plan_type_knowledge
type_copy_of_plan_knowledge
plan_knowledge
physical_goal_knowledge
physical_type_knowledge
type_copy_of_physical_knowledge
physical_knowledge
visual_knowledge

Figure 35: The implemented mind monitoring application, MindMon, shown here with the simple block building domain and the reflective thinking layers described in this dissertation.

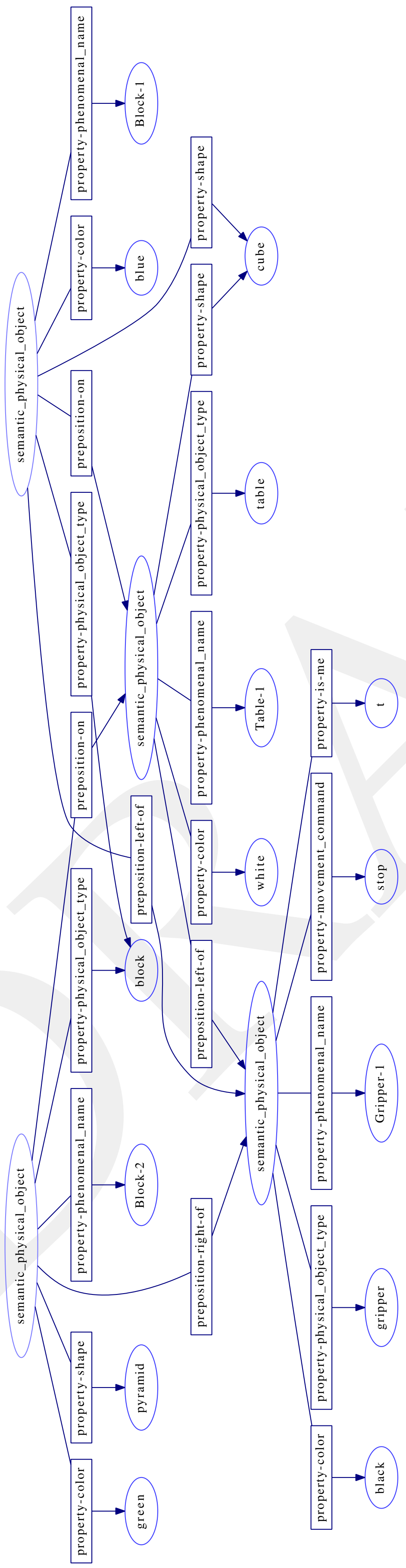


Figure 36: The implemented physical knowledge-base.

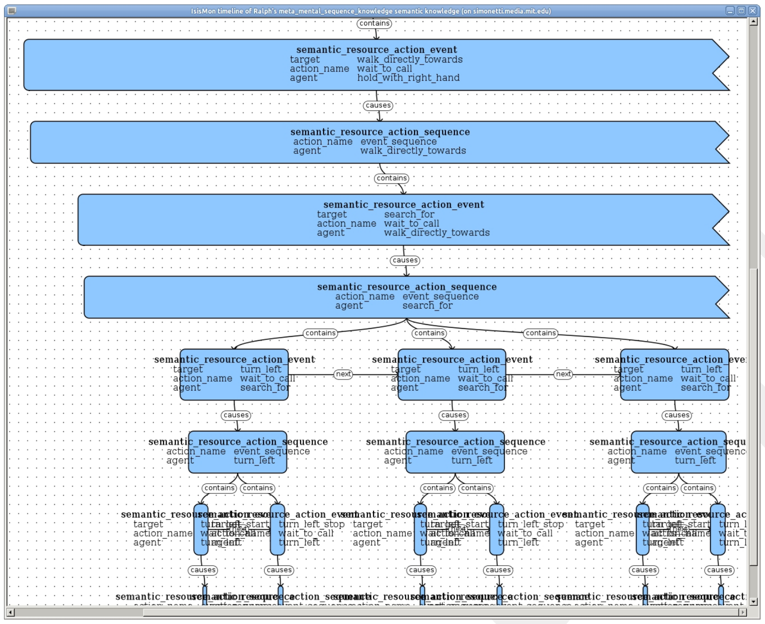

IsisMon timeline of Ralph's meta_mental_sequence_knowledge semantic knowledge (on simonetti.media.mit.edu)
contains
semantic_resource_action_event
target walk_directly_towards
action_name wait_to_call
agent hold_with_right_hand
causes
semantic_resource_action_sequence
action_name event_sequence
agent walk_directly_towards
contains
semantic_resource_action_event
target search_for
action_name wait_to_call
agent walk_directly_towards
causes
semantic_resource_action_sequence
action_name event_sequence
agent search_for
contains
next
semantic_resource_action_event
target turn_left
action_name wait_to_call
agent search_for
semantic_resource_action_sequence
action_name event_sequence
agent turn_left
turn_left_start
turn_left_stop

Figure 37: The implemented semantic event knowledge base.

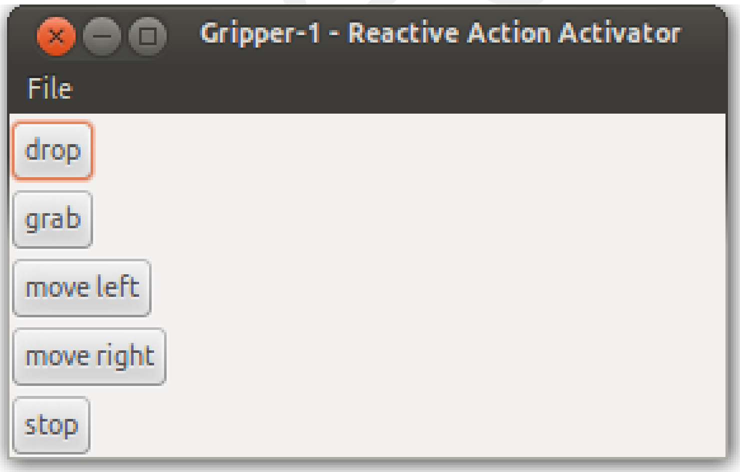


Figure 38: The implemented first-order resource activator.

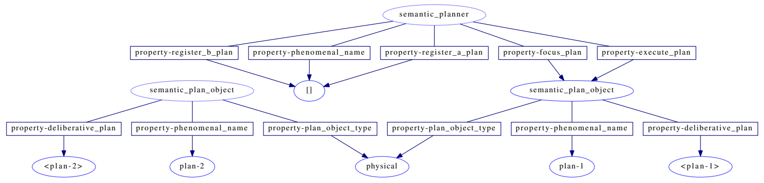
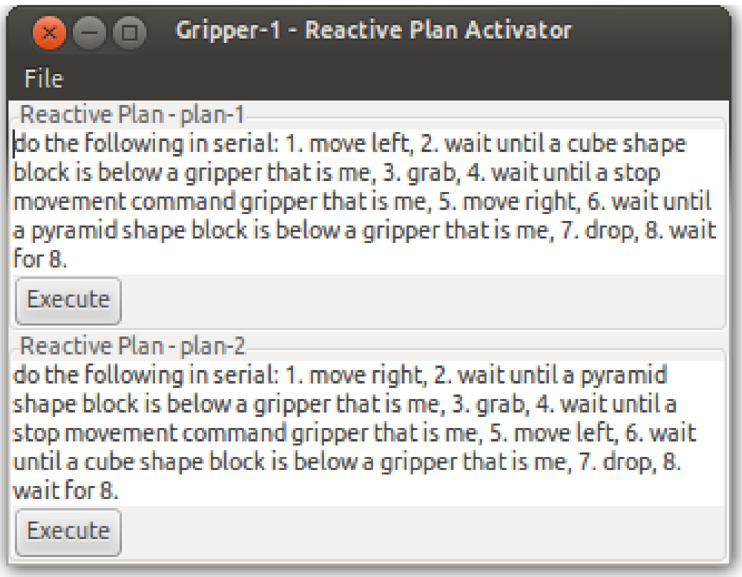


Figure 39: The implemented first-order planning machine knowledge.



do the following in serial: 1. move left, 2. wait until a cube shape block is below a gripper that is me, 3. grab, 4. wait until a stop movement command gripper that is me, 5. move right, 6. wait until a pyramid shape block is below a gripper that is me, 7. drop, 8. wait for 8.

do the following in serial: 1. move right, 2. wait until a pyramid shape block is below a gripper that is me, 3. grab, 4. wait until a stop movement command gripper that is me, 5. move left, 6. wait until a cube shape block is below a gripper that is me, 7. drop, 8. wait for 8.

Figure 40: The implemented first-order plan activator.

17.5 first-order plan activator

Figure 40 shows the implemented first-order plan activator.

17.6 second-order resource activator

Figure 41 shows the implemented second-order resource activator.

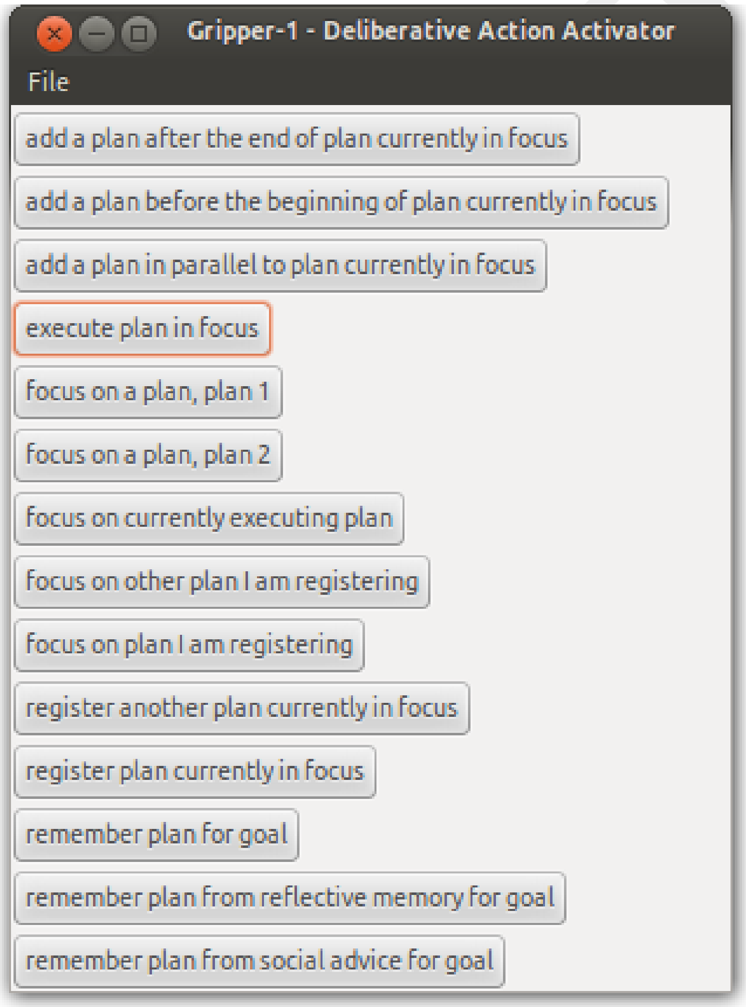

Gripper-1 - Deliberative Action Activator
File
add a plan after the end of plan currently in focus
add a plan before the beginning of plan currently in focus
add a plan in parallel to plan currently in focus
execute plan in focus
focus on a plan, plan 1
focus on a plan, plan 2
focus on currently executing plan
focus on other plan I am registering
focus on plan I am registering
register another plan currently in focus
register plan currently in focus
remember plan for goal
remember plan from reflective memory for goal
remember plan from social advice for goal

Figure 41: The implemented second-order resource activator.

18 RELATED IMPLEMENTATIONS

18.1 HACKER

A good precedent for reflective debugging responses to catalogs of failures is *HACKER*, one of the first reflective planning and debugging models, written by Sussman (1973). [complete]

18.2 EM-ONE

Singh (2005) provided *EM-ONE*, a most recent example of an extension of HACKER that reasons about a more complicated three-dimensional physical and social block building domain. [complete]

Part IV

CONCLUSION

19 EVALUATION

19.1 STREAMING CONSTANT-TIME SYMBOLIZATION

In Part II, I define a symbolic represenation to be a subgraph of the dynamic activity in the layers below the static symbolic reference. In general, this definition becomes a subgraph isomorphism problem. An algorithmic solution to subgraph isomorphism is known to be an NP-complete problem, requiring search. Because my approach requires every activity in the implementation to take a constant, $O(1)$, time, the symbolic perception activity must also be composed of constant time components. The way that this problem is handled in my architecture is by considering the symbolic perception problem as a *planning-to-perceive* problem (Pryor & Collins 1995). In my proof-of-concept implementation, I only use graphs that are of a low enough complexity in order to be bounded by a known constant upper time boundary. My implementation only creates new symbols for those subgraphs that are below the size of four nodes and four edges. I have hard-coded one two types of symbolic perceptions at the second-order reflective level that focuses on plans in the focus and execution registers.

In general, the subgraph isomorphism for specific symbolic perceptions from the layers below becomes a planning problem for each newly created symbol. This allows the planning machine to create plans that are compiled into parallel resources that can note the perception of each newly created symbol, based on the reflective copy of the physical knowledge-base.

20 FUTURE

The reflective model has interesting implications for the future of symbolic approaches to AI. An explicit acknowledgment of symbols being references to the dynamic ongoing activity implies this layered model that I have described.

20.1 PROPOGATING KNOWLEDGE DEPENDENCIES

Propagators, described by Radul & Sussman (2009), could be organized into reflective layers that would allow propagator networks to learn better propagate and merge functions. The main obstacle in combining the propagator model with this reflective model is that the dynamic components of the propagator model must be explicitly represented as knowledge within another propagator model. In a propagator network, this would require that cells contain knowledge learned from watching the execution of propagate and merge functions of other cells. Hypotheses can be learned from watching the propagate and merge functions execute, and these hypotheses can then be propagated in a layer above. Layers of propagator cells allows debugging the representations for how these functions could be changed in order to lead to better ground network performance. In a reflective model, there is a necessary explicit separation between the static and dynamic components of the model. Without strict reflective layers of knowledge, the model becomes tautological as soon as a representation is made of the dynamic component of itself.

20.2 SELF REFLECTION

Viewing the physical activities as physical objects is future research for the model. Thinking about subjective perspectives on physical objects and their relationships is future research in the model.

The ability to abstract object and subject distinctions within the mind is key to how I see the creation of selves. The most important and the first subject and object distinctions are those that help accomplish or avoid physical goals. In creating the distinction that allows the mind to see the physical body as subjected to the objects of a physical world, the mind has created one of the first subjective perspectives, the physical body. An object has subjective perspectives and extents in Space. Further, every subjective perspective would usefully have a set of causal hypotheses that lead to and away from the object's more central subjective perspectives. Objects, therefore, become collections of subjective perspectives that can be easily traversed given the included causal models. The extent of an object in Space gives a useful Spatial arrangement for causal hypotheses that help to create controllable boundaries around similar perceptual and goal activities.

20.3 circular objects

Causal hypotheses are used in order to predict the future occurrence of symbolic perceptions and goals. Through reflective thinking, plans are constructed from causal hypotheses consistent with the past, elaborating the necessities in the past and the results in the future. Analogies between consistent plans can be abstracted into models of objects. For example, a plan that begins and ends with the same symbolic perception could be considered an example of a "circular" object; thus, an analogical plan abstraction would represent a type of object, that has multiple sides to perceive depending on the subject's position in the plan. Circular objects may be an important type of object to analogically recognize because a circular object allows one to perform actions while being able to get back to a known perceptual symbol. If one is in a known circular object, then one is not lost in the sense that one can always follow the circle in order to get back to any perceptual symbol contained within the circular plan object. Although, the implementation includes tools for performing analogical abstractions and constructions of plans, this machinery is not key to the thesis of explaining causal reflective learning to accomplish goals. This section is included in the dissertation to eliminate a potential confusion that would conflate the concept of symbol with that of object. Symbols are the most primitive elements available to thinking, while objects are more complicated in that they are composed of analogical consistencies between plans that are themselves composed of many symbols. Objects have multiple subjective perspectives. Objects are static creations of a mind, based on static symbols representing the ongoing activity in Duration.

20.4 body and world

An intelligent mind builds models that predict the occurrence of perceptual symbols. Analogical thinking is used to abstract objects and simultaneously the implied subjective perspectives on these objects as they are manipulated. The separation between the physical body and the physical world is an objective form of thinking that is fundamentally caused by goals that emphasize distinguishing bodily and worldly goals in the causal structures of the physical perceptions. For example, Bongard et al. (2006) demonstrates an approaches referred to as "motor babbling" that use this approach in a non-reflective model in order to learn a physical self-model. This approach learns about physical actions that change physical perceptions in predictable ways are often bodily physical perceptions, i.e. moving ones hand in front of ones face causes one to consistently see a hand. Also, physical pain perceptions could be related to physical goals that emphasize the static separation between the body and the world. Understanding the static separation of the body and the world allows for an objective form of scientific study that places the object of study in the world outside of the effects of the body.

20.5 m^{th}-stratum self reflection

After discussing strong parallels between my model and Minsky's six-layered reflective model of mind, called The Emotion Machine or Model-6, in section 7.1, I will continue a description of the top two layers of Minsky's model in the terms of my model.

Above this ability to model multiple interacting objects is the ability to model objects as containing objects of the self-same type. The ability to abstract all of the layers of a reflective mind into one type of object or another can be used to recursively model one object's model of another object. In learning to abstract subjective perspectives on objects from the mind, the distinction between the body and the world can be imagined to be one of the most important fundamental subject and object distinctions that would be made by such a system. In my model, abstracting part of the mind to be an object and another part to be a subject is the same as creating a "self" model. In other words, the initial objective abstractions in this model would be a distinction between a subjective self and an objective physical world. Therefore, although my model does not include self reflection, I see this form of reflection as along a new dimension that has, as its origin, all of the reflective layers that I have so far described in my model. I will refer to this new dimension as ordered *strata* of the *self dimension of reflective thinking*.

$$\text{self}^0 \text{ reflective stratum} \approx \bigcup_{n=0}^{\infty} \text{reflective}^n \text{ layer} \tag{20.1}$$

$$\text{self}^m \text{ reflective stratum} \approx \bigcup_{n=0}^{\infty} \text{self}^m \text{ reflective}^n \text{ layer} \tag{20.2}$$

Prior to the first objective thinking stratum existing, it must have a prior part of the mind upon which to objectively reflect. I will refer to all of the reflective layers I've described so far as the zeroth-stratum of objective thinking, the "self^0" stratum, since it contains no objective thinking at all. Therefore, the m^{th}-stratum of objective thinking in my model can be described as "self^m" reflective thinking. Given this notation, the self-reflective layer of Minsky's model would be the first-stratum of objective thinking, or self^1 reflection, in my model. Each self^m reflective stratum in my model contains an infinite number of self^m reflective^n layers. Equations **??** and 20.2, roughly and non-mathematically show how each objective thinking stratum is composed of an infinite number of reflective layers. Again, this notation is *not* set theoretic, and thus, set theoretical mathematics do not apply.

$$\left.\begin{array}{l}\text{Minsky's Problem Domain}\\ \text{Minsky's Built-in Reactive Layer}\\ \text{Minsky's Learned Reactive Layer}\\ \text{Minsky's Deliberative Layer}\\ \text{Minsky's Reflective Layer}\end{array}\right\} \approx \bigcup_{n=0}^{\infty} \text{self}^{0} \text{ reflective}^{n} \text{ layer}$$

$$\approx \text{self}^{0} \text{ reflective stratum}$$

$$\text{Minsky's Self-Reflective Layer} \approx \bigcup_{n=0}^{\infty} \text{self}^{1} \text{ reflective}^{n} \text{ layer}$$

$$\approx \text{self}^{1} \text{ reflective stratum}$$

$$\text{Minsky's Self-Conscious Layer} \approx \bigcup_{n=0}^{\infty} \text{self}^{2} \text{ reflective}^{n} \text{ layer}$$

$$\approx \text{self}^{2} \text{ reflective stratum}$$

Figure 42: The top layers of Model-6 roughly mapped to the suggested self^{m} reflective^{n} stratum notation.

Minsky's "self-conscious" layer would thereby roughly correspond with the self^{2} reflective stratum. Having two levels of recursion in an objective thinking sense, leads to the ability to represent what one object references in terms of what another object references in terms of the original object. Figure 42 shows roughly how the levels of Minsky's model can be related to the suggested stratum notation.

20.6 IMPRIMERS

Minsky (2006) explains that terms like "guilt" and "pride" can refer to self-conscious ways of thinking. Minsky hypothesizes that there are powerful ways of thinking that result when someone whom one respects, like a parent, values or devalues their goals. This type of relationship between object models is named an *imprimer relationship* by Minsky. In this relationship, the parent plays the role of what Minsky has named an *imprimer*, in reference to the ability of ducklings to learn, relatively arbitrarily, to trust and follow a parent-like figure. He hypothesizes that imprimers, such as parents and caregivers, play an important role in unquestioned knowledge inheritance in children. This type of thinking about what someone else is thinking about their goals requires at least two strata of self reflection in my model.

Part V

APPENDIX

A SCIENCE

Physically objectifying problems scientifically has proved useful. The primary utility of this understanding is that the scientist can be replaced by another scientist and the experimental results can be duplicated. Arbitrary replacement of the subjective scientist implies the potential for mechanical simulation of the physical phenomenon, reducing the subjective scientist to a purely perceptual role. Mechanical duplication of human labor leads to increased efficiency and productivity. First, steam engines were a model of autonomous human activity. Next, computers became the physical model of choice. Now, various humanoid robots have included electrical motors and computers to become closer reproductions of human physical abilities. Thus, understanding a model of mind has become important in duplicating more complex human physical behaviors that require thinking. The mechanical duplication of human thinking is the goal of the field of AI. Only the simplest forms of human thinking have been coherently mechanized thus far.

A.1 OBJECT AND SUBJECT

Causal hypotheses are used in order to predict the future occurrence of symbolic perceptions and goals. Through reflective thinking, plans are constructed from causal hypotheses consistent with the past, elaborating the necessities in the past and the results in the future. Analogies between consistent plans can be abstracted into models of objects. For example, a plan that begins and ends with the same symbolic perception could be considered an example of a "circular" object; thus, an analogical plan abstraction would represent a type of object, that has multiple sides to perceive depending on the subject's position in the plan. Circular objects may be an important type of object to analogically recognize because a circular object allows one to perform actions while being able to get back to a known perceptual symbol. If one is in a known circular object, then one is not lost in the sense that one can always follow the circle in order to get back to any perceptual symbol contained within the circular plan object. My implementation includes tools for performing analogical abstractions and constructions of plans, but this machinery is not key to my thesis of explaining causal reflective learning to accomplish goals. I include this section in the dissertation to eliminate a potential confusion that would conflate the concept of symbol with that of object. Symbols are the most primitive elements available to thinking, while objects are more complicated in that they are composed of analogical consistencies between plans that are themselves composed of many symbols. Objects have multiple subjective perspectives. Objects are static creations of a mind, based on static symbols representing the ongoing activity in Duration.

A.2 BODY AND WORLD

An intelligent mind builds models that predict the occurrence of perceptual symbols. Analogical thinking is used to abstract objects and simultaneously the implied subjective perspectives on these objects as they are manipulated. The separation between the physical body and the physical world is an objective form of thinking that is fundamentally caused by goals that emphasize distinguishing bodily and worldly goals in the causal structures of the physical perceptions. For example, physical actions that change physical perceptions in predictable ways are often bodily physical perceptions, i.e. moving ones hand in front of ones face causes one to consistently see a hand. Also, physical pain perceptions could be related to physical goals that emphasize the static separation between the body and the world. Understanding the static separation of the body and the world allows for an objective form of scientific study that places the object of study in the world outside of the effects of the body.

A.3 THE DYNAMIC AS "OUT THERE"

Scientists and others often view the dynamic as "out there". What this means is that the dynamic is viewed as a collection of objects, some of which have been discovered, and many of which are still "out there" and ready to be discovered. "Out there" generally means outside of the physical body and in the physical world. To me, this view of the dynamic as "out there" is grounded on a physically objective view, which implies a subject; equivalently, one could state that this view of the dynamic being "out there" could be grounded on a subjective view, which implies objects. In either case, the view of objects is not mentally grounded on anything less than objects or their implied subjects. The mind is not an object, and, logically, studying the mind is not a subjective task. The scientist studies objects to great utility, but what is often not clear are the goals that have driven the measurement of utility with these subjective objectifications. The objects are taken as given to the mentality as if the mind did not create the objects in order to think subjectively in the first place. The idea of objective thinking being less prone to error than subjective thinking is purely two sides of the same coin, and the error in both cases is in respect to the goals of the creator of the distinction. With every object is an implied set of subjective perspectives, and every object and subject distinction is useful with respect to the goals that relate to its creation and experimentation.

One of the key distinctions between those who study the mind and scientists is that scientists do not explicitly model the goals that drive the creation of their objects of study, while those who model the mind do not model an object from a subjective perspective because the mind is not an object. The mind is everything that exists. A model of the mind includes and explicitly states the goals that drive it to create objects and their implied subjective perspectives. Therefore, the dynamic is not "out there" but the dynamic is everything that exists. Seeing the dynamic as "out there" is based on a static object and subject distinction that often carries with it undisclosed goals for its creation. It is therefore a mistake to see this static object and subject distinction as an unquestionable fundamental quality of existence.

It is an option for scientists to be aware of the goals that drive their subjective studies of objects. A goal is not a physical object that a scientist can study subjectively. A goal is part of a model of mind; so, therefore, in order for a scientist to be aware of their goals, he or she must understand a model of mind that allows them to understand their chosen goals in relation to everything else that also exists. Without this awareness, there is a critical danger of the blind leading the blind with the scientist not being aware of their choice of a subjective perspective on the world.

Understanding this distinction replaces the role of the subjected victim with that of the responsible creator.

Most importantly, we are not fundamentally subjected to objects; we are instead the creators of static objects, based on many static symbolic references to the actual dynamic ongoing activity. This is a subtle distinction, but understanding this distinction replaces a role of the subjected victim of fundamentally unquestionable objects with the role of the personally responsible creator of static objects with implied subjective perspectives. Understanding that objects are the static creations of an intelligent mind is something that seems to have escaped the explicit statement of many great scientists. The fact that the study of the mind is not a subjective field separates it from all sciences. The mind is not able to be studied scientifically for this exact reason. Science is based on objective and subjective distinctions that are creations of a mind. Studying and understanding a model of mind is akin to understanding causality; a prerequisite before posing a scientific hypothesis. Therefore, the dynamic is not "out there" as many scientists currently believe; the dynamic is everything that exists, all activity that is currently ongoing in Duration, the mind.

A.4 PHYSICS

The physicist uses an objective form of thinking in order to separate his or her self as the subject from the object of their study. Objectification of physical perceptions has led to the static creation of universes, galaxies, stars, planets, humans, organs, cells, molecules, atoms, and even the fascinating electrons, which can be thought of as wave objects or particle objects, depending on the goals of the subject. In each of these studies, symbolic perceptions are correlated into causal hypotheses and these causal hypotheses are grouped into plans that are analogically abstracted into objects, each different objective abstraction having different subjective implications. Note that objects and their implied subjective views are static constructions caused by the goals of an intelligent mind. The fact that the position and momentum of an electron can be considered objectively as probability distributions is fundamentally derived from counting symbolic perceptions and putting the resulting numbers into ratios. The symbolic perceptions in this task are the closest thing to the dynamic activity of perception. Once the dynamic activity of perception has been reflectively symbolized, all that remains is to manipulate the static Spatial arrangements of these symbols.

a.5 neuroscience

In the field of AI, there is a tendency to confuse an objective scientific model of the brain with a model of mind, which is, among other things, an explanation of the modelling of the object and subject distinction. Neuroscience is fundamentally an objective physical science. Neurons are one of the most important objective discoveries of neuroscience. Like all objective physical sciences, the objects discovered are caused by goals. The objective study of neuroscience has led to distinctions between the central and peripheral nervous systems; forebrain, midbrain, and hindbrain divisions; cortical columns and circuits; nerves and pathways; all composed of neurons. The objectification of the nervous system is led by the goals of the scientist, which are usually medical. Recently, the study of neuroscience has combined with psychology and AI, which allows for studying not just the physical structure, but how this physical structure relates to a model of mind. How is reflective thinking, which requires symbols, done by the brain? This is an interesting question, but I want to make the point that in order to study a question like this one must not confuse a model of mind with the physical objective model, both being necessary for posing a clear question relating thinking and the brain.

a.6 neural networks

Neural networks are one of the objects created by the field of neuroscience. Given the physically objective assumptions of the field, we have learned of the interactions between many different objects related to neural networks: the brain is composed of approximately one hundred billion neurons; chemical gradients guide neuron growth and attrition; neurons use chemical binding and electrical potential differences to communicate through axons to dendrites. There are many computational models that have been created to explain different physical phenomena that arise from the interactions of many neurons. The study of neural networks is an objective physical science, like all of neuroscience. Neural networks are static objective creation of a mind.

A.7 COMPUTATIONAL NEURAL NETWORKS

Computational models of biological neural networks are generally referred to as *artificial neural networks*. Note that there is a danger of confusion here between the initial static construction of the neural network object and subject distinction in the mind of the scientist and the further static construction of a computational description of the subjective view of this object. Since both of these objects are static by definition, I will use the term *computational neural network* when I am specifically referring to the simulation of the mathematical neural network object.

Computational neural network models were originally created in order to explain the behavior of physical neurons; however, a subset of these models that were modelled as simple linear and non-linear algebraic equations have become popular as a subset of control and optimization model. These control models been found to be useful for controlling robot arms, car braking systems, and even music recommendation systems.

It is important to make a distinction between three ideas here: neural networks, numerical control models, and thinking. Thinking is the activity that builds models of all of these phenomena, including itself. Algebraic control systems are symbolic models that assume that one knows how to count, add, and multiply. Neural networks are an object created by physically objective neuroscience. Because a neural network is a physical object, it is an analogical abstraction of plans composed of causal models that are themselves composed of symbols that refer to physical actualities. Note that of these three ideas, thinking is the only one that is actual; the other two are static models, which are created by the thinking activity.

A.8 BRAIN AND MIND

The brain is one of the key organs that keeps humans alive. The nervous system is a key factor in muscular movement of the body; perceptions, including vision, smell, touch, etc.; speech production; language understanding; thinking, including counting, making plans, and doing mathematical calculations. When I say that the brain is key, what I mean is that when the brain is damaged, these functions are lost. Understanding neural networks, like most neuroscientific pursuits, is often driven by medical goals. If a neural network is understood in terms of how it leads to functionality, such as thinking, then we can use this physical understanding in correlation with a model of mind to design preventative, rehabilitative, and augmentative approaches to neural medicine.

There is a danger at this point of becoming confused and accepting a model in place of the the dynamic of the ongoing activity in Duration, which requires neither symbols, objects, nor any other form of thinking to exist. A model of mind is a construction that is used to think about this ongoing existence. A model of the brain is a construction that is based on the static physically objective assumptions that divide bodies, organs, cells, etc. Note that a model of mind does not necessarily make any objective assumptions; in this sense, a model of mind can model the exploration and creative activity that leads to a model of the brain. A model of the brain is a physically objective model, and does not lead to models of thinking because of the assumptions of purely physical objective focus. However, regardless of focus, both examples of modelling must keep the the dynamic of the ongoing activity as the fundamental referent for any symbolized perceptions or causal models. Some models may be derivatives of others, but all models are static.

B EDUCATION

B.1 SKILL AND UNDERSTANDING

To understand an activity in Duration is to actively maintain a causal model of its effects. There are many activities in Duration that are referred to as skills that do not require understanding. For example, an understanding of walking is not necessary for the skillful activity of walking to occur. Skills are symbolic references to activities that may be refined through symbolic repetition without having any causal model existing for the activity. Also, one may have an understanding without having any symbolic skill referring to the activity. Often, an understanding of the activity can guide a practice routine for refining the skill. Coaches use an understanding of a skill to provide instructions for practicing skills. When skills are practiced, the activity in Duration changes. The understanding of the skill does not necessarily change during the practice. Practicing a skill can simply lead to sub-symbolic changes to the activity that the symbolic skill is a referent.

B.2 COACHING AND TEACHING

Coaching and teaching play similar roles in changing the mind of the disciple. The purpose of coaching is to change sub-symbolic skillful activities, while the purpose of teaching is to give the disciple an understanding of an activity. The roles of coach and teacher work well together. For example, the master may teach the disciple an understanding of a symbolic activity before coaching how to practice the skill. Often, an understanding is used as a crutch for initially learning to practice a skill, and once the disciple has learned to practice the skill, the understanding is no longer necessary and is sometimes even a distraction from its mastery.

B.3 UNDERSTANDING AND MECHANICAL SIMULATION

If an activity is understood, then a mechanical model of the activity can be built, allowing for the symbolic simulation of the activity. Computers allow for the mechanical simulation of the active maintenance of symbolic relationships in Space; computers also allow for the simulation of causal models, allowing transitions from the past to the future. Therefore, computers allow for the mechanical simulation of the understanding of an activity. In this sense, the existence of a mechanical simulation of the understanding of an activity is an existence proof for the understanding of the activity. Further, for any understanding, a mechanical simulation can be built. Therefore, a one-to-one relationship exists between an understanding and a mechanical simulation.

B.4 EDUCATION THROUGH NEUROSCIENCE

Education is a field that is designed to coach and teach an understanding of skillful thinking activities. Current technology uses cleverly designed forms of reading and writing as the primary tool for determining whether or not a child has learned the target skills. As the skills become longer and more complex, it becomes harder for this form of linguistic evaluation to be designed by the teacher. Modern brain scanning technologies have begun to explore exciting new ways to approach education of sub-linguistic symbolic thinking. This opens the possibility for the coaching and teaching of pre-linguistic forms of skillful thinking and understanding.

The design, assignment, and grading of homework for schoolchildren is expensive, time consuming, and error prone, when humans perform these tasks. If parts of these tasks could be automated, more children would become better educated, given the same resources. Because of growing interest in this potential from both an educational and mental health perspective, I here explain the relationship between the physical objective science and the model of mind, which are both necessary for automating these tasks.

When physical phenomena are measured and correlated with a model of mind, a potential exists for mechanically automating a program based on educational goals, emphasizing augmentation and rehabilitation of mental behavior. Valuing mental activities in correlation with physical phenomena, such as neural networks as measured by brain scanning technologies, will lead to a new approach to more efficient and more exact educational tools. Current reliance on symbolic written output from complex thinking tasks is a critical bottleneck in all forms of education and a critical impasse for teaching pre-linguistic forms of thinking.

Children learn in many ways. One way is through their personal experience playing with the physical world. Another important way is through understanding language and inheriting knowledge directly from parents and teachers in a language form. Having a model of mind that explains how causal models are learned and debugged in both of these cases is important for an embracing neuroscientific approach to education.

Having a model of mind that provides explanatory descriptions for many types of learning is important to allow for a wide range of children to be approached sensitively and idiosyncratically with an educational program that helps them at their individual stages of learning. Developing a program that enables students to usefully create, incorporate, and debug knowledge from many different sources is important for developing a sound inherited culture and adaptable individuals.

C THE CODE

C.1 OPEN-SOURCE DOWNLOAD

All of the code that I developed for this PhD is free and openly developed, can be downloaded from my webpage, and compiled by simply typing "./configure; make". See the on-line Funk2 website for download instruction for SALS, http://funk2.org/, for downloads, documentation, user community resources, and latest news. Also, see http://em-two.net/ for research news on my Moral Compass cognitive architecture that currently is distributed with SALS.

C.2 README

```
funk2: causally reflective programming language

  funk2 [-x <command>] [-p <portnum>] [<source.fu2>]

    <source.fu2>

        A user supplied filename of file from which to read and
            execute source
        code after booting and before exiting.

    -x <command>   [:default [repl]]

        A user supplied command to execute after booting and
            before exiting.

    -p <portnum>   [:default 22222 :try anything from 22222 to
        23221]

        The localhost peer-command-server port number.  Each
            copy of funk2
        sharing a network interface must be able to allocate a
            unique
        peer-command-server port number.


TO PERFORM LOCAL BUILD:

  ./configure
  make


TO RUN LOCAL BUILD:

  ./funk2.sh    (from original compile directory)


TO PERFORM SYSTEM-WIDE INSTALLATION:

  ./configure
  make
  make install  (as root)
```

```
TO RUN SYSTEM—WIDE INSTALLATION:

  funk2          (from anywhere on system)



Homepage:

  http://funk2.org/



Git Access:

  http://git.neuromin.de/


Last Modified: 2010.10.25

Code Mass
______________________________________________________________________________


    Lines of Funk2 Code.....:   49837 total
    Words of Funk2 Code.....:  246131 total
    Characters of Funk2 Code: 2516032 total

    Lines of C Code.........:  131529 total
    Words of C Code.........:  521743 total
    Characters of C Code....: 6982294 total

    Total Lines of Code.....:  182371 total
    Total Words of Code.....:  770823 total
    Total Characters of Code: 9553461 total

______________________________________________________________________________


README Last Generated: Mon Aug  1 14:05:18 EDT 2011
```

C.3 FILE LISTING

```
start_emacs.sh
README
configure.ac
Makefile.am
funk—mode.el
c/configurator.c
c/debugbreak.c
c/f2_agent.c
c/f2_agent.h
c/f2_ansi.c
c/f2_ansi.h
c/f2_apropos.c
c/f2_apropos.h
c/f2_archconfig.h
c/f2_array.c
c/f2_array.h
c/f2_atomic.h
c/f2_buffered_file.c
c/f2_buffered_file.h
c/f2_buffered_socket.c
c/f2_buffered_socket.h
c/f2_bug.c
c/f2_bug.h
c/f2_bytecodes.c
c/f2_bytecodes.h
c/f2_cause.c
```

```
c/f2_cause.h
c/f2_chunk.c
c/f2_chunk.h
c/f2_circular_buffer.c
c/f2_circular_buffer.h
c/f2_command_line.c
c/f2_command_line.h
c/f2_compile.c
c/f2_compile.h
c/f2_compile_x86.c
c/f2_compile_x86.h
c/f2_core_extension.c
c/f2_core_extension_funk.c
c/f2_core_extension_funk.h
c/f2_core_extension.h
c/f2_cpu.c
c/f2_cpu.h
c/f2_debug_macros.h
c/f2_dlfcn.c
c/f2_dlfcn.h
c/f2_dptr.c
c/f2_dptr.h
c/f2_dynamic_memory.c
c/f2_dynamic_memory.h
c/f2_f2ptr.c
c/f2_f2ptr.h
c/f2_fiber.c
c/f2_fiber.h
c/f2_fileio.c
c/f2_fileio.h
c/f2_frame_objects.c
c/f2_frame_objects.h
c/f2_funk2_node.c
c/f2_funk2_node.h
c/f2_funktional.c
c/f2_funktional.h
c/f2_garbage_collector.c
c/f2_garbage_collector.h
c/f2_garbage_collector_pool.c
c/f2_garbage_collector_pool.h
c/f2_globalenv.c
c/f2_globalenv.h
c/f2_global.h
c/f2_glwindow.c
c/f2_glwindow.h
c/f2_gmodule.c
c/f2_gmodule.h
c/f2_graph.c
c/f2_graph_cluster.c
c/f2_graph_cluster.h
c/f2_graph.h
c/f2_graph_match_error_correcting.c
c/f2_graph_match_error_correcting.h
c/f2_graphviz.c
c/f2_graphviz.h
c/f2_gtk.c
c/f2_gtk.h
c/f2_hash.c
c/f2_hash.h
c/f2_html.c
c/f2_html.h
c/f2_knowledge.c
c/f2_knowledge.h
c/f2_larva.c
c/f2_larva.h
c/f2_load.c
c/f2_load.h
c/f2_malloc.c
c/f2_malloc.h
c/f2_management_thread.c
c/f2_management_thread.h
c/f2_matlab.c
```

```
c/f2_matlab.h
c/f2_memblock.c
c/f2_memblock.h
c/f2_memory.c
c/f2_memory.h
c/f2_memorypool.c
c/f2_memorypool.h
c/f2_module_registration.c
c/f2_module_registration.h
c/f2_natural_language.c
c/f2_natural_language.h
c/f2_never_delete_list.c
c/f2_never_delete_list.h
c/f2_nil.c
c/f2_nil.h
c/f2_number.c
c/f2_number.h
c/f2_object.c
c/f2_object.h
c/f2_opengl.c
c/f2_opengl.h
c/f2_optimize.c
c/f2_optimize.h
c/f2_package.c
c/f2_package.h
c/f2_package_handler.c
c/f2_package_handler.h
c/f2_packet.c
c/f2_packet.h
c/f2_partial_order.c
c/f2_partial_order.h
c/f2_peer_command_server.c
c/f2_peer_command_server.h
c/f2_perception_lattice.c
c/f2_perception_lattice.h
c/f2_primes.c
c/f2_primes.h
c/f2_primfunks.c
c/f2_primfunks__errno.c
c/f2_primfunks__errno.h
c/f2_primfunks__fcntl.c
c/f2_primfunks__fcntl.h
c/f2_primfunks.h
c/f2_primfunks__ioctl.c
c/f2_primfunks__ioctl.h
c/f2_primfunks__locale.c
c/f2_primfunks__locale.h
c/f2_primfunks__string.c
c/f2_primfunks__string.h
c/f2_primobject__boolean.c
c/f2_primobject__boolean.h
c/f2_primobject__char_pointer.c
c/f2_primobject__char_pointer.h
c/f2_primobject__circular_buffer.c
c/f2_primobject__circular_buffer.h
c/f2_primobject__doublelinklist.c
c/f2_primobject__doublelinklist.h
c/f2_primobject__dynamic_library.c
c/f2_primobject__dynamic_library.h
c/f2_primobject__environment.c
c/f2_primobject__environment.h
c/f2_primobject__fiber_trigger.c
c/f2_primobject__fiber_trigger.h
c/f2_primobject__file_handle.c
c/f2_primobject__file_handle.h
c/f2_primobject__frame.c
c/f2_primobject__frame.h
c/f2_primobject__hash.c
c/f2_primobject__hash.h
c/f2_primobject__interval_tree-bak.c
c/f2_primobject__interval_tree-bak.h
c/f2_primobject__largeinteger.c
```

```
c/f2_primobject__largeinteger.h
c/f2_primobject__list.c
c/f2_primobject__list.h
c/f2_primobject__matrix.c
c/f2_primobject__matrix.h
c/f2_primobject__object.c
c/f2_primobject__object.h
c/f2_primobject__object_type.c
c/f2_primobject__object_type.h
c/f2_primobject__ptypehash.c
c/f2_primobject__ptypehash.h
c/f2_primobject__redblacktree.c
c/f2_primobject__redblacktree.h
c/f2_primobjects.c
c/f2_primobject__set.c
c/f2_primobject__set.h
c/f2_primobjects.h
c/f2_primobject__stream.c
c/f2_primobject__stream.h
c/f2_primobject__tensor.c
c/f2_primobject__tensor.h
c/f2_primobject__traced_cmutex.c
c/f2_primobject__traced_cmutex.h
c/f2_primobject_type.c
c/f2_primobject_type.h
c/f2_primobject_type_handler.c
c/f2_primobject_type_handler.h
c/f2_print.c
c/f2_print.h
c/f2_processor.c
c/f2_processor.h
c/f2_processor_mutex.c
c/f2_processor_mutex.h
c/f2_processor_thread.c
c/f2_processor_thread.h
c/f2_processor_thread_handler.c
c/f2_processor_thread_handler.h
c/f2_protected_alloc_array.c
c/f2_protected_alloc_array.h
c/f2_ptype.c
c/f2_ptype.h
c/f2_ptypes.c
c/f2_ptypes.h
c/f2_ptypes_memory.h
c/f2_reader.c
c/f2_reader.h
c/f2_redblacktree.c
c/f2_redblacktree.h
c/f2_reflective_memory.c
c/f2_scheduler.c
c/f2_scheduler.h
c/f2_scheduler_thread_controller.c
c/f2_scheduler_thread_controller.h
c/f2_search.c
c/f2_search.h
c/f2_set.c
c/f2_set.h
c/f2_signal.c
c/f2_signal.h
c/f2_simple_repl.c
c/f2_simple_repl.h
c/f2_socket.c
c/f2_socket_client.c
c/f2_socket_client.h
c/f2_socket.h
c/f2_socket_server.c
c/f2_socket_server.h
c/f2_sort.c
c/f2_sort.h
c/f2_staticmemory.c
c/f2_staticmemory.h
c/f2_status.c
```

```
c/f2_status.h
c/f2_string.c
c/f2_string.h
c/f2_surrogate_parent.c
c/f2_surrogate_parent.h
c/f2_system_file_handler.c
c/f2_system_file_handler.h
c/f2_system_headers.h
c/f2_system_processor.c
c/f2_system_processor.h
c/f2_terminal_print.c
c/f2_terminal_print.h
c/f2_termios.c
c/f2_termios.h
c/f2_time.c
c/f2_time.h
c/f2_trace.c
c/f2_trace.h
c/f2_tricolor_set.c
c/f2_tricolor_set.h
c/f2_user_thread_controller.c
c/f2_user_thread_controller.h
c/f2_virtual_processor.c
c/f2_virtual_processor.h
c/f2_virtual_processor_handler.c
c/f2_virtual_processor_handler.h
c/f2_virtual_processor_thread.c
c/f2_virtual_processor_thread.h
c/f2_xmlrpc.c
c/f2_xmlrpc.h
c/f2_zlib.c
c/f2_zlib.h
c/funk2.c
c/funk2.h
c/funk2_main.c
c/test.c
fu2/action.fu2
fu2/actor.fu2
fu2/actortest.fu2
fu2/assembler.fu2
fu2/bootstrap-apropos.fu2
fu2/bootstrap-array.fu2
fu2/bootstrap-boot.fu2
fu2/bootstrap-bug.fu2
fu2/bootstrap-bugs.fu2
fu2/bootstrap-cause.fu2
fu2/bootstrap-compound_object.fu2
fu2/bootstrap-cons.fu2
fu2/bootstrap-core_extension.fu2
fu2/bootstrap-core_extension_funk.fu2
fu2/bootstrap-critic.fu2
fu2/bootstrap-critics-reactive.fu2
fu2/bootstrap-default_critics.fu2
fu2/bootstrap-define_bug_responses.fu2
fu2/bootstrap-dynamic_library.fu2
fu2/bootstrap-fiber.fu2
fu2/bootstrap-frame.fu2
fu2/bootstrap.fu2
fu2/bootstrap-garbage_collector.fu2
fu2/bootstrap-graph.fu2
fu2/bootstrap-graph-old.fu2
fu2/bootstrap-grid.fu2
fu2/bootstrap-hash.fu2
fu2/bootstrap-largeinteger.fu2
fu2/bootstrap-list.fu2
fu2/bootstrap-object.fu2
fu2/bootstrap-package.fu2
fu2/bootstrap-primobject.fu2
fu2/bootstrap-ptypes.fu2
fu2/bootstrap-reader.fu2
fu2/bootstrap-redblacktree.fu2
fu2/bootstrap-repl.fu2
```

```
fu2/bootstrap-set_theory.fu2
fu2/bootstrap-sort.fu2
fu2/bootstrap-string.fu2
fu2/bootstrap-surrogate_parent.fu2
fu2/bootstrap-terminal_print.fu2
fu2/bootstrap-type_conversions.fu2
fu2/bootstrap-zlib.fu2
fu2/brainviz.fu2
fu2/cardgame-ai.fu2
fu2/cardgame.fu2
fu2/cause.fu2
fu2/characters.fu2
fu2/compile.fu2
fu2/emailcharacters.fu2
fu2/emotionmachine.fu2
fu2/english-eval.fu2
fu2/graph.fu2
fu2/graph_match_test.fu2
fu2/graphviz.fu2
fu2/internet.fu2
fu2/link-grammar-wrapper.fu2
fu2/miscfunks.fu2
fu2/neuralmom-brain_area.fu2
fu2/neuralmom-demo.fu2
fu2/neuralmom-nervous_system.fu2
fu2/neuralmom-occipital_cortex.fu2
fu2/opengl.fu2
fu2/pattern.fu2
fu2/planner.fu2
fu2/primfunks-apropos.fu2
fu2/primfunks-arithmetic.fu2
fu2/primfunks-array.fu2
fu2/primfunks-bug.fu2
fu2/primfunks-cause.fu2
fu2/primfunks-chunk.fu2
fu2/primfunks-compile.fu2
fu2/primfunks-core_extension.fu2
fu2/primfunks-core_extension_funk.fu2
fu2/primfunks-cpu.fu2
fu2/primfunks-dlfcn.fu2
fu2/primfunks-errno.fu2
fu2/primfunks-fcntl.fu2
fu2/primfunks-fiber.fu2
fu2/primfunks-fiber_trigger.fu2
fu2/primfunks-frame.fu2
fu2/primfunks.fu2
fu2/primfunks-garbage_collector.fu2
fu2/primfunks-gmodule.fu2
fu2/primfunks-graph.fu2
fu2/primfunks-hash.fu2
fu2/primfunks-ioctl.fu2
fu2/primfunks-largeinteger.fu2
fu2/primfunks-locale.fu2
fu2/primfunks-management_thread.fu2
fu2/primfunks-object.fu2
fu2/primfunks-optimize.fu2
fu2/primfunks-package.fu2
fu2/primfunks-package_handler.fu2
fu2/primfunks-primes.fu2
fu2/primfunks-primobjects.fu2
fu2/primfunks-primobject_type.fu2
fu2/primfunks-primobject_type_handler.fu2
fu2/primfunks-print.fu2
fu2/primfunks-ptypes.fu2
fu2/primfunks-reader.fu2
fu2/primfunks-redblacktree.fu2
fu2/primfunks-scheduler.fu2
fu2/primfunks-search.fu2
fu2/primfunks-set.fu2
fu2/primfunks-socket.fu2
fu2/primfunks-sort.fu2
fu2/primfunks-string.fu2
```

```
fu2/primfunks—surrogate_parent.fu2
fu2/primfunks—terminal_print.fu2
fu2/primfunks—termios.fu2
fu2/primfunks—time.fu2
fu2/primfunks—trace.fu2
fu2/primfunks—virtual_processor_handler.fu2
fu2/primfunks—zlib.fu2
fu2/reactive.fu2
fu2/readline—wrapper.fu2
fu2/repl.fu2
fu2/rlglue—wrapper.fu2
fu2/serialize.fu2
fu2/story.fu2
fu2/thought_process.fu2
fu2/trace.fu2
fu2/x86—compile.fu2
fu2/x86—compile—machine_code.fu2
fu2/x86—compile—mov.fu2
misc/fables.fu2
misc/frog_and_toad.fu2
misc/frog—and—toad.fu2
misc/officer_joke.fu2
misc/roboverse—blocks_world.fu2
misc/roboverse—demo.fu2
misc/simple_game.fu2
extension/cairo/cairo.c
extension/cairo/cairo.h
extension/causality/causality.c
extension/conceptnet/conceptnet.c
extension/conceptnet/conceptnet.h
extension/concept_version_space/concept_version_space.c
extension/concept_version_space/concept_version_space.h
extension/equals_hash/equals_hash.c
extension/equals_hash/equals_hash.h
extension/event_stream/event_stream.c
extension/event_stream/event_stream.h
extension/fibermon/fibermon.c
extension/forgetful_event_stream/forgetful_event_stream.c
extension/forgetful_event_stream/forgetful_event_stream.h
extension/forward_planner/forward_planner.c
extension/frame_ball/frame_ball.c
extension/graph_isomorphism/graph_isomorphism.c
extension/graph_isomorphism/graph_isomorphism.h
extension/image/image.c
extension/image/image.h
extension/image_sequence/image_sequence.c
extension/image_sequence/image_sequence.h
extension/interval_tree/interval_tree.c
extension/interval_tree/interval_tree.h
extension/lick/lick.c
extension/lick/lick.h
extension/mentality/mentality.c
extension/mentality/mentality.h
extension/meta_semantic_knowledge_base/meta_semantic_knowledge_
    base.c
extension/meta_semantic_knowledge_base/meta_semantic_knowledge_
    base.h
extension/movie/movie.c
extension/movie/movie.h
extension/propogator/propogator.c
extension/propogator/propogator.h
extension/semantic_action_event/semantic_action_event.c
extension/semantic_action_event/semantic_action_event.h
extension/semantic_agent/semantic_agent.c
extension/semantic_agent/semantic_agent.h
extension/semantic_category/semantic_category.c
extension/semantic_category/semantic_category.h
extension/semantic_causal_event/semantic_causal_event.c
extension/semantic_causal_event/semantic_causal_event.h
extension/semantic_causal_object/semantic_causal_object.c
extension/semantic_causal_object/semantic_causal_object.h
```

```
extension/semantic_containment_object/semantic_containment_
      object.c
extension/semantic_containment_object/semantic_containment_
      object.h
extension/semantic_directed_action_event/semantic_directed_
      action_event.c
extension/semantic_directed_action_event/semantic_directed_
      action_event.h
extension/semantic_event_knowledge_base/semantic_event_knowledge
      _base.c
extension/semantic_event_knowledge_base/semantic_event_knowledge
      _base.h
extension/semantic_event/semantic_event.c
extension/semantic_event/semantic_event.h
extension/semantic_event_sequence/semantic_event_sequence.c
extension/semantic_event_sequence/semantic_event_sequence.h
extension/semantic_event_tree/semantic_event_tree.c
extension/semantic_event_tree/semantic_event_tree.h
extension/semantic_frame/semantic_frame.c
extension/semantic_frame/semantic_frame.h
extension/semantic_goal_action_causal_hypothesis/semantic_goal_
      action_causal_hypothesis.c
extension/semantic_goal_action_causal_hypothesis/semantic_goal_
      action_causal_hypothesis.h
extension/semantic_goal/semantic_goal.c
extension/semantic_goal/semantic_goal.h
extension/semantic_knowledge_base/semantic_knowledge_base.c
extension/semantic_knowledge_base/semantic_knowledge_base.h
extension/semantic_know_of_existence_event/semantic_know_of_
      existence_event.c
extension/semantic_know_of_existence_event/semantic_know_of_
      existence_event.h
extension/semantic_know_of_relationship_event/semantic_know_of_
      relationship_event.c
extension/semantic_know_of_relationship_event/semantic_know_of_
      relationship_event.h
extension/semantic_object/semantic_object.c
extension/semantic_object/semantic_object.h
extension/semantic_object_type/semantic_object_type.c
extension/semantic_object_type/semantic_object_type.h
extension/semantic_ordered_object/semantic_ordered_object.c
extension/semantic_ordered_object/semantic_ordered_object.h
extension/semantic_packable_object/semantic_packable_object.c
extension/semantic_packable_object/semantic_packable_object.h
extension/semantic_physical_object/semantic_physical_object.c
extension/semantic_physical_object/semantic_physical_object.h
extension/semantic_physical_object_type_relation/semantic_
      physical_object_type_relation.c
extension/semantic_physical_object_type_relation/semantic_
      physical_object_type_relation.h
extension/semantic_physical_object_type/semantic_physical_object
      _type.c
extension/semantic_physical_object_type/semantic_physical_object
      _type.h
extension/semantic_proprioception/semantic_proprioception.c
extension/semantic_proprioception/semantic_proprioception.h
extension/semantic_proprioceptual_object/semantic_proprioceptual
      _object.c
extension/semantic_proprioceptual_object/semantic_proprioceptual
      _object.h
extension/semantic_proprioceptual_orientation/semantic_
      proprioceptual_orientation.c
extension/semantic_proprioceptual_orientation/semantic_
      proprioceptual_orientation.h
extension/semantic_proprioceptual_position/semantic_
      proprioceptual_position.c
extension/semantic_proprioceptual_position/semantic_
      proprioceptual_position.h
extension/semantic_realm/semantic_realm.c
extension/semantic_realm/semantic_realm.h
extension/semantic_relationship_key/semantic_relationship_key.c
extension/semantic_relationship_key/semantic_relationship_key.h
```

```
extension/semantic_resource_action_event/semantic_resource_
     action_event.c
extension/semantic_resource_action_event/semantic_resource_
     action_event.h
extension/semantic_resource_action_sequence/semantic_resource_
     action_sequence.c
extension/semantic_resource_action_sequence/semantic_resource_
     action_sequence.h
extension/semantic_resource_event_knowledge_base/semantic_
     resource_event_knowledge_base.c
extension/semantic_resource_event_knowledge_base/semantic_
     resource_event_knowledge_base.h
extension/semantic_resource/semantic_resource.c
extension/semantic_resource/semantic_resource.h
extension/semantic_self/semantic_self.c
extension/semantic_self/semantic_self.h
extension/semantic_situation_category/semantic_situation_
     category.c
extension/semantic_situation_category/semantic_situation_
     category.h
extension/semantic_situation/semantic_situation.c
extension/semantic_situation/semantic_situation.h
extension/semantic_situation_transition/semantic_situation_
     transition.c
extension/semantic_situation_transition/semantic_situation_
     transition.h
extension/semantic_somatosensation/semantic_somatosensation.c
extension/semantic_somatosensation/semantic_somatosensation.h
extension/semantic_somatosensory_object/semantic_somatosensory_
     object.c
extension/semantic_somatosensory_object/semantic_somatosensory_
     object.h
extension/semantic_temporal_object/semantic_temporal_object.c
extension/semantic_temporal_object/semantic_temporal_object.h
extension/semantic_time/semantic_time.c
extension/semantic_time/semantic_time.h
extension/semantic_visual_object/semantic_visual_object.c
extension/semantic_visual_object/semantic_visual_object.h
extension/timeline/timeline.c
extension/timeline/timeline.h
built-in/alien/alien.fu2
built-in/ansi/primfunks-ansi.fu2
built-in/basic_bug_responses/basic_bug_responses.fu2
built-in/graph_cluster/bootstrap-graph_cluster.fu2
built-in/graph_cluster/primfunks-graph_cluster.fu2
built-in/graph_match_error_correcting/graph_match_error_
     correcting.fu2
built-in/graph_match_error_correcting/graph_match_error_
     correcting-primfunks.fu2
built-in/graphviz/graphviz.fu2
built-in/graphviz/graphviz-primfunks.fu2
built-in/gtk/bootstrap-gtk.fu2
built-in/gtk/primfunks-gtk.fu2
built-in/math/math.fu2
built-in/mutex/mutex.fu2
built-in/natural_language/dictionary_frame.fu2
built-in/natural_language/natural_language_command.fu2
built-in/natural_language/natural_language-primfunks.fu2
built-in/natural_language/parse_tree.fu2
built-in/natural_language/skb-test.fu2
built-in/number/bootstrap-number.fu2
built-in/number/primfunks-number.fu2
built-in/object_lattice/bootstrap-object_lattice.fu2
built-in/object_lattice/primfunks-object_lattice.fu2
built-in/perception_lattice/bootstrap-perception_lattice.fu2
built-in/perception_lattice/primfunks-perception_lattice.fu2
built-in/utilities/errno.fu2
built-in/utilities/fcntl.fu2
built-in/utilities/ioctl.fu2
built-in/utilities/socket.fu2
built-in/xmlrpc/bootstrap-xmlrpc.fu2
built-in/xmlrpc/primfunks-xmlrpc.fu2
```

```
example/blocks_world/blocks_world_block.fu2
example/blocks_world/blocks_world.fu2
example/blocks_world/blocks_world_gripper_controller.fu2
example/blocks_world/blocks_world_gripper.fu2
example/blocks_world/blocks_world_physics.fu2
example/blocks_world/blocks_world_sprite.fu2
example/blocks_world/blocks_world_window.fu2
example/divisi2/divisi2.fu2
example/em_two_webpage/em_two_webpage.fu2
example/english_language/english_dictionary.fu2
example/english_language/english_dictionary_parse.fu2
example/funk2-htmldoc/funk2-htmldoc.fu2
example/funk2-webpage/funk2-webpage.fu2
example/graph_match/graph_match.fu2
example/graph_match/graph_match-test.fu2
example/gtk_timeline/gtk_timeline.fu2
example/isismon/isismon_agent.fu2
example/isismon/isismon_builtin_reactive_physical_activator.fu2
example/isismon/isismon_deliberative_forward_planner_activator.
    fu2
example/isismon/isismon_deliberative_goal_activator.fu2
example/isismon/isismon.fu2
example/isismon/isismon_knowledge.fu2
example/isismon/isismon_learned_reactive_physical_activator.fu2
example/isis_world_client/isis_world_client.fu2
example/isis_world_demo/isis_agent_body.fu2
example/isis_world_demo/isis_visual_agent.fu2
example/isis_world_demo/isis_visual_object.fu2
example/isis_world_demo/isis_world_demo.fu2
example/isis_world_demo/isis_world.fu2
example/isis_world_demo/jj.fu2
example/little_carol_world/little_carol_world.fu2
example/macbeth/macbeth.fu2
example/mind/agency.fu2
example/mind/agent_body.fu2
example/mind/character.fu2
example/mind/mental_layer.fu2
example/mind/mind.fu2
example/mindmon-1.0/mindmon-1.0.fu2
example/mindmon-blocks_world/mindmon-blocks_world.fu2
example/mindmon-isis_world/mindmon-isis_world-builtin_reactive_
    physical_activator.fu2
example/mindmon-isis_world/mindmon-isis_world-deliberative_goal_
    activator.fu2
example/mindmon-isis_world/mindmon-isis_world.fu2
example/mindmon-isis_world/mindmon-isis_world-learned_reactive_
    physical_activator.fu2
example/mindmon/mindmon_agent.fu2
example/mindmon/mindmon_agent_tool.fu2
example/mindmon/mindmon_agent_tool_widget.fu2
example/mindmon/mindmon_agent_widget.fu2
example/mindmon/mindmon.fu2
example/mindmon/mindmon_knowledge.fu2
example/mindmon/mindmon_world.fu2
example/mind/physical_world.fu2
example/mind/resource.fu2
example/mind/self_model.fu2
example/mind/story.fu2
example/mind/story-graph.fu2
example/moral_compass-isis_world/moral_compass-isis_world-
    builtin_reactive_physical_agency_resources.fu2
example/moral_compass-isis_world/moral_compass-isis_world.fu2
example/moral_compass-isis_world/moral_compass-isis_world-
    learned_reactive_physical_agency_resources.fu2
example/moral_compass-isis_world/moral_compass-isis_world-
    learned_reactive_physical_agency_resources-functions.fu2
example/moral_compass/moral_agent_body.fu2
example/moral_compass/moral_compass.fu2
example/moral_compass/old_reflective_event_knowledge_agency-bak.
    fu2
example/moral_compass/self_conscious_imprimer_agency.fu2
example/moral_compass/self_conscious_layer.fu2
```

```
example/moral_compass/self_reflective_layer.fu2
example/moral_compass/self_reflective_other_agents_knowledge_
      agency.fu2
example/muddy_carol/muddy_carol.fu2
example/rct_webpage/rct_webpage.fu2
example/reflective_mind-blocks_world/reflective_mind-blocks_
      world.fu2
example/reflective_mind/builtin_reactive_layer.fu2
example/reflective_mind/builtin_reactive_neural_plug_agency.fu2
example/reflective_mind/builtin_reactive_physical_agency.fu2
example/reflective_mind/builtin_reactive_sensory_agency.fu2
example/reflective_mind/deliberative_execution_agency.fu2
example/reflective_mind/deliberative_forward_planner_agency.fu2
example/reflective_mind/deliberative_forward_planner_agency-test
      .fu2
example/reflective_mind/deliberative_goal_creation_agency.fu2
example/reflective_mind/deliberative_goal_event_knowledge_agency
      .fu2
example/reflective_mind/deliberative_goal_knowledge_agency.fu2
example/reflective_mind/deliberative_layer.fu2
example/reflective_mind/deliberative_physical_event_knowledge_
      agency.fu2
example/reflective_mind/learned_reactive_language_agency.fu2
example/reflective_mind/learned_reactive_layer.fu2
example/reflective_mind/learned_reactive_physical_agency.fu2
example/reflective_mind/learned_reactive_physical_knowledge_
      agency.fu2
example/reflective_mind/learned_reactive_sensory_agency.fu2
example/reflective_mind/reflective_credit_assignment_agency.fu2
example/reflective_mind/reflective_event_knowledge_agency.fu2
example/reflective_mind/reflective_layer.fu2
example/reflective_mind/reflective_mind.fu2
example/reflective_mind/reflective_mind_perception.fu2
example/reflective_mind/reflective_mind_proprioceptual_object.fu
      2
example/reflective_mind/reflective_mind_visual_agent.fu2
example/reflective_mind/reflective_mind_visual_object.fu2
example/roboverse/roboverse.fu2
example/socket-client/socket-client.fu2
example/socket-server/socket-server.fu2
example/traced_mind/traced_mind.fu2
example/traced_mind/traced_resource.fu2
example/visualize/isismon_agent_visualization.fu2
example/visualize/visualize_test.fu2
extension/cairo/cairo-core.fu2
extension/conceptnet/conceptnet-core.fu2
extension/concept_version_space/concept_version_space-core.fu2
extension/equals_hash/equals_hash-core.fu2
extension/event_stream/event_stream-core.fu2
extension/fibermon/fibermon1.fu2
extension/fibermon/fibermon-core.fu2
extension/fibermon/fibermon.fu2
extension/forgetful_event_stream/forgetful_event_stream-core.fu2
extension/forward_planner/forward_planner-core.fu2
extension/frame_ball/frame_ball-core.fu2
extension/graph_isomorphism/graph_isomorphism-core.fu2
extension/image/image-core.fu2
extension/image_sequence/image_sequence-core.fu2
extension/interval_tree/interval_tree-core.fu2
extension/lick/lick-core.fu2
extension/mentality/mentality-core.fu2
extension/meta_semantic_knowledge_base/meta_semantic_knowledge_
      base-core.fu2
extension/movie/movie-core.fu2
extension/propogator/propogator-core.fu2
extension/semantic_action_event/semantic_action_event-core.fu2
extension/semantic_agent/semantic_agent-core.fu2
extension/semantic_category/semantic_category-core.fu2
extension/semantic_causal_event/semantic_causal_event-core.fu2
extension/semantic_causal_object/semantic_causal_object-core.fu2
extension/semantic_containment_object/semantic_containment_
      object-core.fu2
```

```
extension/semantic_directed_action_event/semantic_directed_
      action_event—core.fu2
extension/semantic_event_knowledge_base/semantic_event_knowledge
      _base—core.fu2
extension/semantic_event/semantic_event—core.fu2
extension/semantic_event_sequence/semantic_event_sequence—core.
      fu2
extension/semantic_event_tree/semantic_event_tree—core.fu2
extension/semantic_frame/semantic_frame—core.fu2
extension/semantic_goal_action_causal_hypothesis/semantic_goal_
      action_causal_hypothesis—core.fu2
extension/semantic_goal/semantic_goal—core.fu2
extension/semantic_knowledge_base/semantic_knowledge_base—core.
      fu2
extension/semantic_know_of_existence_event/semantic_know_of_
      existence_event—core.fu2
extension/semantic_know_of_relationship_event/semantic_know_of_
      relationship_event—core.fu2
extension/semantic_object/semantic_object—core.fu2
extension/semantic_object_type/semantic_object_type—core.fu2
extension/semantic_ordered_object/semantic_ordered_object—core.
      fu2
extension/semantic_packable_object/semantic_packable_object—core
      .fu2
extension/semantic_physical_object/semantic_physical_object—core
      .fu2
extension/semantic_physical_object_type_relation/semantic_
      physical_object_type_relation—core.fu2
extension/semantic_physical_object_type/semantic_physical_object
      _type—core.fu2
extension/semantic_proprioception/semantic_proprioception—core.
      fu2
extension/semantic_proprioceptual_object/semantic_proprioceptual
      _object—core.fu2
extension/semantic_proprioceptual_orientation/semantic_
      proprioceptual_orientation—core.fu2
extension/semantic_proprioceptual_position/semantic_
      proprioceptual_position—core.fu2
extension/semantic_realm/semantic_realm—core.fu2
extension/semantic_relationship_key/semantic_relationship_key—
      core.fu2
extension/semantic_resource_action_event/semantic_resource_
      action_event—core.fu2
extension/semantic_resource_action_sequence/semantic_resource_
      action_sequence—core.fu2
extension/semantic_resource_event_knowledge_base/semantic_
      resource_event_knowledge_base—core.fu2
extension/semantic_resource/semantic_resource—core.fu2
extension/semantic_self/semantic_self—core.fu2
extension/semantic_situation_category/semantic_situation_
      category—core.fu2
extension/semantic_situation/semantic_situation—core.fu2
extension/semantic_situation_transition/semantic_situation_
      transition—core.fu2
extension/semantic_somatosensation/semantic_somatosensation—core
      .fu2
extension/semantic_somatosensory_object/semantic_somatosensory_
      object—core.fu2
extension/semantic_temporal_object/semantic_temporal_object—core
      .fu2
extension/semantic_time/semantic_time—core.fu2
extension/semantic_visual_object/semantic_visual_object—core.fu2
extension/timeline/timeline—core.fu2
test/cairo—test/cairo—test.fu2
test/concept_version_space—test/concept_version_space—test.fu2
test/gtk—test/gtk—test.fu2
test/interval_tree—test/interval_tree—test.fu2
test/optimize—test/optimize—test.fu2
test/propogator—test/propogator—test.fu2
test/timeline—test/timeline—test.fu2
test/xmlrpc—test/xmlrpc—test.fu2
built—in/alien/alien.fpkg
```

```
built-in/ansi/ansi.fpkg
built-in/basic_bug_responses/basic_bug_responses.fpkg
built-in/graph_cluster/graph_cluster.fpkg
built-in/graph_match_error_correcting/graph_match_error_
    correcting.fpkg
built-in/graphviz/graphviz.fpkg
built-in/gtk/gtk.fpkg
built-in/math/math.fpkg
built-in/mutex/mutex.fpkg
built-in/natural_language/natural_language.fpkg
built-in/number/number.fpkg
built-in/object_lattice/object_lattice.fpkg
built-in/perception_lattice/perception_lattice.fpkg
built-in/utilities/utilities.fpkg
built-in/xmlrpc/xmlrpc.fpkg
example/blocks_world/blocks_world.fpkg
example/divisi2/divisi2.fpkg
example/em_two_webpage/em_two_webpage.fpkg
example/english_language/english_language.fpkg
example/funk2-htmldoc/funk2-htmldoc.fpkg
example/funk2-webpage/funk2-webpage.fpkg
example/graph_match/graph_match.fpkg
example/graph_match/graph_match-test.fpkg
example/gtk_timeline/gtk_timeline.fpkg
example/isismon/isismon.fpkg
example/isis_world_client/isis_world_client.fpkg
example/isis_world_demo/isis_world_demo.fpkg
example/little_carol_world/little_carol_world.fpkg
example/macbeth/macbeth.fpkg
example/mind/mind.fpkg
example/mindmon-1.0/mindmon-1.0.fpkg
example/mindmon-blocks_world/mindmon-blocks_world.fpkg
example/mindmon-isis_world/mindmon-isis_world.fpkg
example/mindmon/mindmon.fpkg
example/moral_compass-isis_world/moral_compass-isis_world.fpkg
example/moral_compass/moral_compass.fpkg
example/muddy_carol/muddy_carol.fpkg
example/rct_webpage/rct_webpage.fpkg
example/reflective_mind-blocks_world/reflective_mind-blocks_
    world.fpkg
example/reflective_mind/reflective_mind.fpkg
example/roboverse/roboverse.fpkg
example/socket-client/socket-client.fpkg
example/socket-server/socket-server.fpkg
example/traced_mind/traced_mind.fpkg
example/visualize/isismon_agent_visualization.fpkg
example/visualize/visualize_test.fpkg
extension/cairo/cairo.fpkg
extension/conceptnet/conceptnet.fpkg
extension/concept_version_space/concept_version_space.fpkg
extension/equals_hash/equals_hash.fpkg
extension/event_stream/event_stream.fpkg
extension/fibermon/fibermon.fpkg
extension/forgetful_event_stream/forgetful_event_stream.fpkg
extension/forward_planner/forward_planner.fpkg
extension/frame_ball/frame_ball.fpkg
extension/graph_isomorphism/graph_isomorphism.fpkg
extension/image/image.fpkg
extension/image_sequence/image_sequence.fpkg
extension/interval_tree/interval_tree.fpkg
extension/lick/lick.fpkg
extension/mentality/mentality.fpkg
extension/meta_semantic_knowledge_base/meta_semantic_knowledge_
    base.fpkg
extension/movie/movie.fpkg
extension/propogator/propogator.fpkg
extension/semantic_action_event/semantic_action_event.fpkg
extension/semantic_agent/semantic_agent.fpkg
extension/semantic_category/semantic_category.fpkg
extension/semantic_causal_event/semantic_causal_event.fpkg
extension/semantic_causal_object/semantic_causal_object.fpkg
```

```
extension/semantic_containment_object/semantic_containment_
      object.fpkg
extension/semantic_directed_action_event/semantic_directed_
      action_event.fpkg
extension/semantic_event_knowledge_base/semantic_event_knowledge
      _base.fpkg
extension/semantic_event/semantic_event.fpkg
extension/semantic_event_sequence/semantic_event_sequence.fpkg
extension/semantic_event_tree/semantic_event_tree.fpkg
extension/semantic_frame/semantic_frame.fpkg
extension/semantic_goal_action_causal_hypothesis/semantic_goal_
      action_causal_hypothesis.fpkg
extension/semantic_goal/semantic_goal.fpkg
extension/semantic_knowledge_base/semantic_knowledge_base.fpkg
extension/semantic_know_of_existence_event/semantic_know_of_
      existence_event.fpkg
extension/semantic_know_of_relationship_event/semantic_know_of_
      relationship_event.fpkg
extension/semantic_object/semantic_object.fpkg
extension/semantic_object_type/semantic_object_type.fpkg
extension/semantic_ordered_object/semantic_ordered_object.fpkg
extension/semantic_packable_object/semantic_packable_object.fpkg
extension/semantic_physical_object/semantic_physical_object.fpkg
extension/semantic_physical_object_type_relation/semantic_
      physical_object_type_relation.fpkg
extension/semantic_physical_object_type/semantic_physical_object
      _type.fpkg
extension/semantic_proprioception/semantic_proprioception.fpkg
extension/semantic_proprioceptual_object/semantic_proprioceptual
      _object.fpkg
extension/semantic_proprioceptual_orientation/semantic_
      proprioceptual_orientation.fpkg
extension/semantic_proprioceptual_position/semantic_
      proprioceptual_position.fpkg
extension/semantic_realm/semantic_realm.fpkg
extension/semantic_relationship_key/semantic_relationship_key.
      fpkg
extension/semantic_resource_action_event/semantic_resource_
      action_event.fpkg
extension/semantic_resource_action_sequence/semantic_resource_
      action_sequence.fpkg
extension/semantic_resource_event_knowledge_base/semantic_
      resource_event_knowledge_base.fpkg
extension/semantic_resource/semantic_resource.fpkg
extension/semantic_self/semantic_self.fpkg
extension/semantic_situation_category/semantic_situation_
      category.fpkg
extension/semantic_situation/semantic_situation.fpkg
extension/semantic_situation_transition/semantic_situation_
      transition.fpkg
extension/semantic_somatosensation/semantic_somatosensation.fpkg
extension/semantic_somatosensory_object/semantic_somatosensory_
      object.fpkg
extension/semantic_temporal_object/semantic_temporal_object.fpkg
extension/semantic_time/semantic_time.fpkg
extension/semantic_visual_object/semantic_visual_object.fpkg
extension/timeline/timeline.fpkg
test/cairo-test/cairo-test.fpkg
test/concept_version_space-test/concept_version_space-test.fpkg
test/gtk-test/gtk-test.fpkg
test/interval_tree-test/interval_tree-test.fpkg
test/optimize-test/optimize-test.fpkg
test/propogator-test/propogator-test.fpkg
test/timeline-test/timeline-test.fpkg
test/xmlrpc-test/xmlrpc-test.fpkg
example/visualize/framework.py
example/visualize/globalvars.py
example/visualize/graphics.py
example/visualize/graphics_testing.py
example/visualize/physics.py
example/visualize/vector.py
example/visualize/visualization.py
```

```
example/visualize/xmlclient.py
example/visualize/xmlserver.py
python/funk2module/setup.py
```

Bergson, H. (1910), 'Time and free will, trans', *F. Pogson. London: Swan Sonnenschein* .

Berkeley, G. (1734), *A treatise concerning the principles of human knowledge*, Jacob Tonson.

Bongard, J., Zykov, V. & Lipson, H. (2006), 'Resilient machines through continuous self-modeling', *Science* **314**(5802), 1118–1121.

Cox, M. (2007), Metareasoning, monitoring, and self-explanation, *in* 'Proceedings of the First International Workshop on Metareasoning in Agent-based Systems, AAMAS-07', pp. 46–60.

Cox, M. & Raja, A. (2008), Metareasoning: A manifesto, *in* 'Proceedings of AAAI 2008 Workshop on Metareasoning: Thinking about Thinking', pp. 1–4.

Dennett, D. (2009), 'The Evolution of Confusion', lecture given at AAI, t.33:25, Technical report.

Fikes, R. & Nilsson, N. (1972), 'Strips: A new approach to the application of theorem proving to problem solving', *Artificial intelligence* **2**(3-4), 189–208.

Graham, J. L. (1969), 'Bass for Sly and the Family Stone: Stand!'.

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

Johnson, L. (1984), 'Bass for Stanley Clarke: Time Exposure'.

Kaelbling, L., Littman, M. & Moore, A. (1996), 'Reinforcement learning: A survey', *Arxiv preprint cs.AI/9605103* .

Maes, P. (1988), Issues in computational reflection, *in* 'Meta-level architectures and reflection', North-Holland, pp. 21–35.

Magritte, R. (1919), The treason of images (1928-9). ("this is not a pipe.") marcel duchamp, lhooq, Technical report.

Messmer, B. (1995), 'Efficient graph matching algorithms'.

Minsky, M. (2005), 'Internal Grounding'.

Minsky, M. (2006), *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*, Simon & Schuster, New York, New York.

Mitchell, T. (1997), 'Machine learning (ise)', *Recherche* **67**, 02.

Pryor, L. & Collins, G. (1995), Planning to Perceive, *in* A. Ram & D. Leake, eds, 'Goal-driven learning', MIT Press, Cambridge, Massachusetts, chapter 10, pp. 287–296.

Radul, A. & Sussman, G. (2009), The art of the propagator, Technical report, Technical Report MIT-CSAIL-TR-2009-002, MIT Computer Science and Artificial Intelligence Laboratory.

Singh, P. (2005), EM-ONE: An Architecture for Reflective Commonsense Thinking, PhD thesis, Massachusetts Institute of Technology.

Singh, P. & Minsky, M. (2003), 'An architecture for combining ways to think', *Proceedings of the International Conference on Knowledge Intensive Multi-Agent Systems* .

Singh, P. & Minsky, M. (2005), An architecture for cognitive diversity, *in* D. Davis, ed., 'Visions of Mind', Idea Group Inc., London.

Smith, D. & Morgan, B. (2010), Isisworld: An open source commonsense simulator for ai researchers, *in* 'Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence'.

Sussman, G. (1973), A computational model of skill acquisition, PhD thesis, Massachusetts Institute of Technology.

Winston, P. (1970), Learning structural descriptions from examples, PhD thesis, Massachusetts Institute of Technology.

COLOPHON

This thesis was typeset with $\LaTeX 2_\varepsilon$ 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 2012

Bo Morgan