MIT-CSAIL-TR-2004-040
AITR-2004-004

June 16, 2004

Building Grounded Abstractions for Artificial Intelligence Programming

Robert A. Hearn

Building Grounded Abstractions for Artificial Intelligence Programming

by

Robert A. Hearn

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2001

Certified by: Gerald J. Sussman
Matsushita Professor of Electrical Engineering
Thesis Supervisor

Accepted by: Arthur C. Smith
Chairman, Department Committee on Graduate Students

Building Grounded Abstractions for Artificial Intelligence Programming

by
Robert A. Hearn

Submitted to the Department of Electrical Engineering and Computer Science on May, 11, 2001, in partial fulfillment of the requirements for the degree of Master of Science

Abstract

Most Artificial Intelligence (AI) work can be characterized as either "high-level" (e.g., logical, symbolic) or "low-level" (e.g., connectionist networks, behavior-based robotics). Each approach suffers from particular drawbacks. High-level AI uses abstractions that often have no relation to the way real, biological brains work. Low-level AI, on the other hand, tends to lack the powerful abstractions that are needed to express complex structures and relationships. I have tried to combine the best features of both approaches, by building a set of programming abstractions defined in terms of simple, biologically plausible components. At the "ground level", I define a primitive, perceptron-like computational unit. I then show how more abstract computational units may be implemented in terms of the primitive units, and show the utility of the abstract units in sample networks. The new units make it possible to build networks using concepts such as long-term memories, short-term memories, and frames. As a demonstration of these abstractions, I have implemented a simulator for "creatures" controlled by a network of abstract units. The creatures exist in a simple 2D world, and exhibit behaviors such as catching mobile prey and sorting colored blocks into matching boxes. This program demonstrates that it is possible to build systems that can interact effectively with a dynamic physical environment, yet use symbolic representations to control aspects of their behavior.

Thesis Supervisor: Gerald J. Sussman
Title: Matsushita Professor of Electrical Engineering

Acknowledgments

I would like to thank my advisor Gerry Sussman for his support and valuable suggestions.

I would like to thank Joel Moses for additional support.

I would like to thank Marvin Minsky, Jake Beal, Push Singh, and Jeremy Zucker for many useful discussions.

I would also like to thank Marvin Minsky for his beautiful Society of Mind model of how the mind works. This thesis is based on those ideas.

Finally, I would like to thank my wife Liz Hearn for dragging me kicking and screaming to Massachusetts.

Contents

Chapter 1

Introduction

In this chapter I describe my objectives in this work, and explain what I mean by "grounded abstractions". I also lay out a road map for the following chapters.

1.1 High-level vs. Low-level AI

There have been two broad divisions of models used for Artificial Intelligence programming. The traditional, "high-level" approach, which includes symbolic and logical AI, has been to specify programs at a very abstract level. Alternatives such as connectionism, neural nets, and behavior-based robotics I refer to as "low-level", because they try to simulate some theory of what real brains actually do at something close to the neural level, without any explicit representational abstractions.

The high-level approach is characterized by the use of powerful programming abstractions, such as procedural representations, arbitrary recursion, and indefinitely large data structures. One drawback of this approach is that the abstractions used often have no relation to the way real, human brains work. Such abstractions may be unjustified, in that there can be unwarranted assumptions that they make sense in the context of a biological brain. In addition, the nature of these abstractions heavily influences the structure of programs written with them. Therefore, programmers are not naturally led to the kinds of solutions to problems that the brain uses.

It is, of course, possible that there are other ways to design an intelligent system than by imitating nature. However, if the ultimate task of AI is to build an artificial human-equivalent intelligence, then it makes sense to be inspired by the solution nature has found. It may not be the only approach, but it is surely worthy of investigation.

The low-level approach is motivated by just such considerations. However, this approach tends to suffer from the reverse problem from high-level AI. Programs or robots created in this paradigm typically lack any explicit abstractions whatsoever, relying instead on the assumption that complex skills such as reasoning and language arise from combinations of simple behaviors. Of course this idea should be explored further, but it is difficult to imagine all of these skills existing without some high-level, symbolic structure. And in any case, without abstractions, it is difficult to design these complex behaviors.

1.2 Grounded Abstractions

I have tried to combine the best features of both of these approaches, removing their weaknesses, by building a set of useful abstractions which are well-grounded in terms of a biologically plausible model. By this I mean that the meanings of the abstractions should be reducible to operations that can plausibly be performed by biological computers, i.e., brains.

I specifically do *not* mean that AI systems need be "grounded" in the sense of requiring literal physical embodiment in a robot. There may be valid sociological reasons for exploring robot intelligence – for example, the fascinating question of what degree of intelligence observers impute to a robot vs. a simulation running the same software. And of course there are pragmatic, commercial reasons for building robots. But there is no compelling reason to believe that intelligence requires a physical body that can interact with the same, real objects that humans interact with. As Marvin Minsky says, "Minds are simply what brains do." [13] The hardware is irrelevant; the process is what matters.

The question of whether intelligence requires a *simulated* environment at all similar to ours is a much more difficult question, which I will not attempt to answer here. Instead, I accept as a reasonable hypothesis the idea that perception and action may contribute important elements to intelligence.

The kinds of systems I propose building have a connectionist flavor, in that they involve simulating networks of interacting units of some sort. Unlike most connectionist systems, the individual units may have a great deal of computational power – trying to build a useful brain by simulating some theoretical model of neurons directly is a daunting task. But the computational capabilities of an individual unit should be of a nature that is reducible to the kinds of operations we can realistically envision actual "meatware" performing.

I take the position that the critical ingredient missing from current AI work is effective models of memory representation and management, and particularly that concepts along the lines of Minsky's *frames* and *K-lines* are essential [11, 12].

What those concepts themselves are lacking is detail. I show how to build systems with frames and K-line-like memories from the ground up, by determining how they can work in terms of elementary computational units, and only gradually throwing away low-level verisimilitude and constructing valid high-level abstractions.

Of course, one can program in terms of memories and frames in a more conventional manner than I am proposing, by simply using the ordinary computer science concepts of variables and records with named fields. However, one should be careful about so cavalierly mapping those cognitive concepts to the nearest computer science equivalent. When designing any system, be it a LISP program, a behavior-based robot, or a space shuttle, the architecture of the system is profoundly influenced by the nature of the tools at hand. The requirements brains impose on how memories can be manipulated may be nothing like the requirements compilers impose on how variables can be manipulated. This leads to putative AI systems which function nothing like the way brains do.

1.3 Learning

Learning is a hallmark of intelligence, and certainly an important requirement for any human-level artificial intelligence. However, I propose that before studying learning as such, we need a firmer grasp on the nature of the underlying representations that are to be modified as learning occurs, and on what the desired results of learning should be. Only after having designed a system capable of performing a certain kind of task will one be able to design a mechanism for learning the appropriate structure.

It is instructive to reflect on the fact that a great deal of what is called intelligence relies not on learning per se, but on the overall memory patterns that are partially the result of learning. It takes on the order of half an hour for the brain to make a permanent memory trace [13]. Thus, any intelligent behavior a person exhibits on a time scale less than this is being performed using only short-term memories.

Theoretically, it should be possible to build a system that behaves like a person, but does not learn. It would be as intelligent as any person, save the ability to form long-term memories. It would be able to solve complex problems, engage in conversations, and interact normally with a complex physical environment – just like the well-known human patient H. M., who is unable to form new long-term memories, but is otherwise quite normal [5, 17].

Of course, one cannot expect to design, by hand, a system with all the knowledge a human has. Some researchers are undertaking the vast, distributed task of accumulating "commonsense knowledge databases", that can serve to replace the knowledge base that humans have had laboriously to learn [7]. This is one possible solution to the problem, but I feel that ultimately most of the knowledge an AI needs will have to be learned – at least once. This is because it seems unlikely to me that the knowledge representations being currently used in such databases will happen to align usefully with the representations that are ultimately required by a truly intelligent machine.

But even if learning is ultimately required, one must still first determine what kinds of structures need to be learned. Therefore, I do not speculate in any detail on unit network learning mechanisms, and the system I implemented does not learn by forming permanent changes to the network structure. I do, however, consider the requirements of long-term memory formation when proposing memory architectures.

1.4 Inspiration and Related Work

Marvin Minsky's "Society of Mind" (SoM) model of how the mind works [13] has been my primary inspiration; I will show how my constructions can be interpreted in the light of that model. In a sense, SoM is an existing model which satisfies my goal of combining the best attributes of high-level and low-level AI. It contains many powerful abstractions, such as frames, polynemes, K-lines, pronomes, etc. Furthermore, unlike most high-level AI, these abstractions are all biologically inspired, and are plausible mechanisms for activities in real, human brains. However, the vast scope of the model means that it necessarily lacks a certain amount of specificity. That is, there is no prescription for actually *implementing* versions of K-lines, frames, etc. Instead, the ideas are presented at the levels of abstraction at which it is easiest to think of them. But without a firm grounding in terms of explicit proposed mechanisms, that one can actually build and test, it is hard to know precisely what the abstractions mean.

This thesis represents a step in the direction of implementing the abstractions described in SoM.

1.5 Overview

In Chapter 2, I introduce my basic computational unit, and suggest some general organizing principles for connecting units together.

In Chapter 3, I describe a mechanism for representing long-term memories, and show how it can be used to connect units together without excessively multiplying the number of wires required.

In Chapter 4, I give an architecture for short-term memories, which are like long-term memories, except that they can be formed with no rewiring required. I show how one may use them to build some useful behaviors that are not typically found in behavior-based robots.

In Chapter 5, I discuss the concept of frame, and show how short-term memories are useful for dealing with frames and frame systems. I propose a novel way of activating and recognizing frames, which puts frames on a more equal footing with other kinds of units.

In Chapter 6, I relate my constructions to various Society of Mind concepts.

In Chapter 7, I describe a simulation which implements the abstractions presented, in the context of a two-dimensional world inhabited by creatures controlled by networks of generalized units.

Chapter 8 summarizes my contributions.

Chapter 2

Units and Agencies

In this chapter I define characteristics of a simple computational unit, which I will use as the basis for further constructions and abstractions.

2.1 Basic Computational Units

My basic computational unit is a simple device which has a number of inputs and outputs, and a current numerical activation level determined by a nonnegative transfer function applied to a weighted sum of the inputs. Units are wired to other units. A weight may be negative; in this case the input is inhibitory (Figure 2.1).

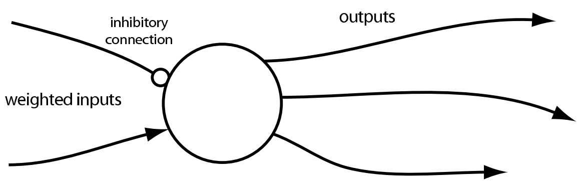


Figure 2.1: A Basic Unit

This kind of unit is clearly inspired by models of neurons [9]. However, I make no attempt to match units directly to neurons, or units to cluster of neurons, or neurons to clusters of units. Neither will I pretend to construct networks that are organized in direct correspondence with the detailed architecture of the brain. Those are tasks for systems neuroscience. Rather, I am after a general organizational compatibility with the way brains could conceivably work.

The primary design constraint in wiring units together is that networks of these units should be organized into structures that are derivable from evolvability considerations – that is, structures that could form by replication and differentiation.

I intentionally leave the dynamics of networks of units somewhat vague. I assume that all the units in a network are continuously updating their activation levels based on their inputs, and that there is some finite propagation delay of signals on the wires. But I make no timing assumptions beyond this when designing constructions in terms of units.

My intention is that all the constructions I make and behaviors I build should be realizable, ultimately, as networks of these basic units. Of course, computers are ultimately realized as vast networks of logic gates, which can be built from these units, so in a sense any program is realizable as a network of my units. However, the human brain does not seem to be organized in the same way as a microprocessor. The abstractions and constructions I build in terms of my units will be motivated by an assumption of biological plausibility.

Some units are taken to be sensors, in that their activation level is a function of interaction with an external environment. These might include tactile, proprioceptive, and visual sensors.

Likewise, some units are actuators, whose computed activation level modifies aspects of the external environment. These would include motors, muscles, etc. It also makes sense to consider actuators which modify the unit network itself, as an underlying mechanism for learning.

2.2 Unit Abstractions

Throughout this thesis I will give definitions of new kinds of units in terms of networks of preexisting units. As a simple example, here is an implementation of an AND-gate as a unit with two inputs, whose transfer function is a step function with a threshold of 1.5:

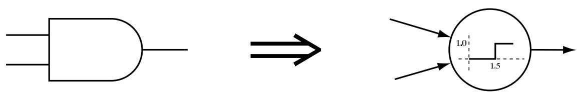


Figure 2.2: AND-gate Abstraction

The symbol '⇒' should be read as "is realized as". Arrows on inputs and outputs will often be omitted when they are clear from context, as they are for logic gates.

Of course, this unit will only function correctly as an AND-gate if its inputs are either 0 or 1, so it is important to be aware of usage constraints when combining units. Such “interconnect standards” are typical of logic families such as TTL.

Other logic gates may be constructed similarly.

2.3 Agencies

I will call a functionally related group of units an *agency*. The units in an agency are assumed to be localized when one thinks of them as existing in a space. This may seem irrelevant when thinking of unit networks as merely describing abstract data flow patterns. But thinking of them as localized helps make unit constructions more biologically plausible, since the components in brains may be constrained in their connection patterns due to varying physical location.

Following on the idea of an agency as a localized clump of units, it is natural to think of the agency as connecting to other agencies via large bundles of wires coming in and going out (Figure 2.3). (The bundles in the figure are displayed schematically; for real agencies there could be hundreds or thousands of wires in a bundle.)

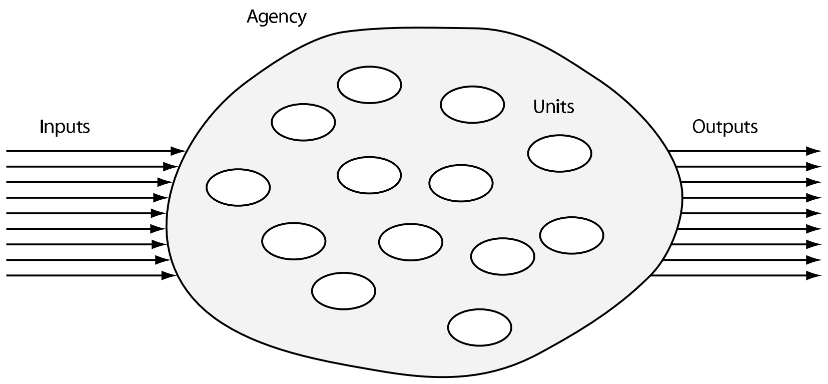


Figure 2.3: An Agency

The behavior of an agency is therefore determined both by its internal connection pattern and state, and by its external environment, as reflected by the input signals.

One can also imagine agencies existing in hierarchies or heterarchies; that is, higher levels of structure can be represented in terms of agency relationships (Figure 2.4).

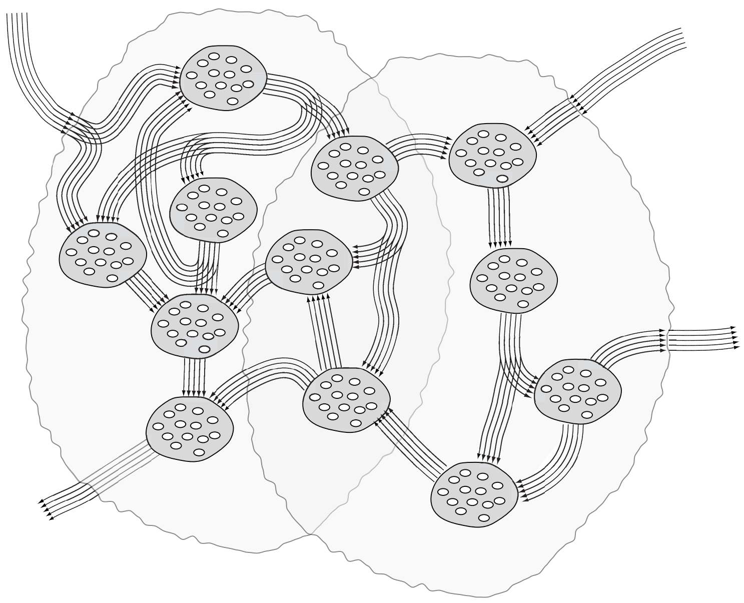


Figure 2.4: Some Possible Agency Relationships

One may conceive of qualitatively different kinds of agencies designed for different purposes. For example, one agency could contain a unit network designed to implement the high-level control processes for finding and eating food. This agency would have a rich internal structure, with groups of units designed to activate in particular patterns and sequences when the appropriate conditions obtained.

Other agencies might serve simply as representations of some detected conditions. For example, an agency could have a set of units, subsets of which are used to denote kinds of object shape or texture. This agency need have no internal structure at all except perhaps for control mechanisms to map between input/output patterns and internal state. These control mechanisms are what I call memories, described in Chapter 3.

2.4 Cross-exclusion

One common pattern of internal connections in an agency involves cross-exclusionary links (Figure 2.5). These ensure that only one unit or related group of units can be active at a time, by using inhibitory connections between the units to be exclusively active. (The inhibitory weights must be greater than 1 for cross-exclusion to work; the higher the weight, the greater the tendency to resist changing state.)

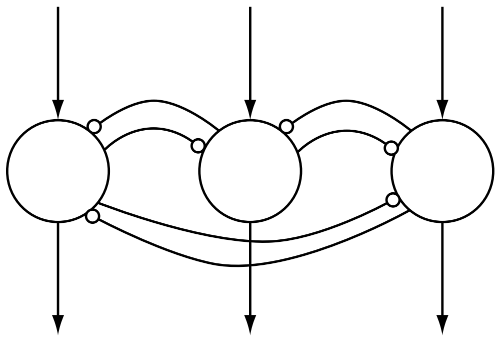


Figure 2.5: Cross-Exclusion

For example, an agency representing color information might cross-exclude units used to represent shades of red from those used to represent shades of blue. Conflicting inputs would lead to one state or another becoming active, but not both. Cross-exclusion is also useful for managing activation of behaviors that have competing resource needs.

Chapter 3

Long-Term Memories

In this chapter I show how to represent long-term memories in unit networks, how to use them to give the networks a degree of plasticity, and how to abstract the constructions for them so that the low-level details do not need to be simulated.

3.1 Activators and Recognizers

An agency is always in some total state determined by the states (activation levels) of its component units.

It is often useful to be able to restore a previous partial or total state of an agency. This is done with memories. Long-term memories may be viewed as I/O units responsible for translating signals on the agencies' interface wires into previously remembered states, and vice-versa. A simple model for a long-term memory *activator* is as follows: it is a unit that detects a particular pattern on the input wires, defined by a particular subset being active at some level, and when thus activated in turn activates some subset of the units in the agency.

Likewise, a long-term memory *recognizer* is a unit that detects a particular activation pattern within the agency, and propagates its activation to some subset of the output wires. The unit activation patterns can correspond to individual units, or to arbitrary subsets of the agency's units.

One can view a memory activation or recognition pattern as a *symbol* denoting a particular partial state of the agency. Note that a set of input/output wires can potentially carry many more symbols than there are wires, by re-using wires among multiple symbols. If there are

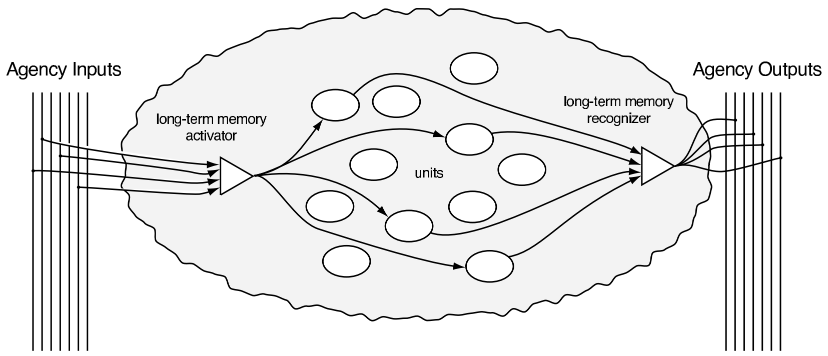


Figure 3.1: Long-Term Memory

enough wires, then multiple symbols may be present on them at once, with minimal interference.

Conceivably the activator and recognizer mechanisms could be bypassed entirely, with the input/output wires mapped directly to individual unit inputs/outputs, one-to-one. However, this would require more interface wires than the scheme described above. It would also have disadvantages regarding the utility of long-term memories, discussed below (Section 3.2, Polynemes).

This coding scheme allows potential connections between large numbers of units, without requiring a dedicated wire for each connection. It was invented by Calvin Mooers in 1947 [16].

In order to make new long-term memories, new units must be allocated and connected to the correct interface wires and internal units. This operation clearly has some analog in biological learning – one form of long-term memory storage appears to be implemented by "long-term potentiation", which modifies synaptic weights [17].

It is desirable to let the network itself, rather than some external mechanism, control the formation of new memories. That is, one would like to be able to build subsystems within the network that detect when relevant events have occurred, and remember useful features of them, by creating new long-term memories. Therefore it seems necessary to extend the model with a mechanism for allocating new units from a pool, and creating or modifying connection weights between units within an agency.

This is easily done – simply assume there is a particular type of actuator unit, the effect of whose activity is to increase or decrease the connection weight between a pair of units it is connected to. The units to be allocated would then be preexisting components in a pool of units with zero connection weights. Such a mechanism does not need to be specified in any detail, because whatever the details, one may treat the mechanism abstractly when programming unit network simulations (Section 3.3, Abstracting Long-Term Memories).

In contrast to typical connectionist models, we don't propose any local learning rules for modifying connection weights. Indeed, we assume that learning is a process carried out at a higher level, by agencies responsible for maintaining other agencies' memories.

3.2 Polynemes

A useful way to use long-term memories and symbols is to induce several agencies to respond simultaneously in their own ways to the same symbol. (This is why long-term memory activators are needed – without them, there would be no way for the same input pattern to be meaningful to multiple agencies.) Minsky calls this a *polyneme*, and I will use his terminology. For example, suppose an agency is used to denote the concept of *apple*, by being put into a particular state. Doing so will activate a specific pattern on the agency's output wires. If these reach several other agencies – say those for representing *shape*, *size*, *color*, etc. – then each such agency can put itself into a state representing the appropriate and respective apple attributes. *Shape* would be in a *round* state, and so on (Figure 3.2).

With polynemes, a collection of individual attributes can be "chunked" into a higher-level symbol used to refer to them collectively. Agencies that want to do something involving apple memories can invoke them with a single symbol instead of one for each attribute-specific agency.

Of course, it would also be useful to be able to recognize the *apple* condition from the conjunction of the attribute-representing agencies being in the appropriate states. If their outputs combine to form part of the inputs to the apple-representing agency, then it can recognize the conjunction of the (generally different) output symbols each attribute agency generates when apple attributes are active. In effect, a new *apple* symbol has been constructed out of the combination of the symbols for *red*, *round*, and so on. This way, when an apple is seen, and the attribute-representing agencies are put into states representing apple attributes (*round*, *red*, *fist-sized*, ...), their long-term memory recognizers output the symbols that the apple polyneme recognizer associates with the *apple* state.

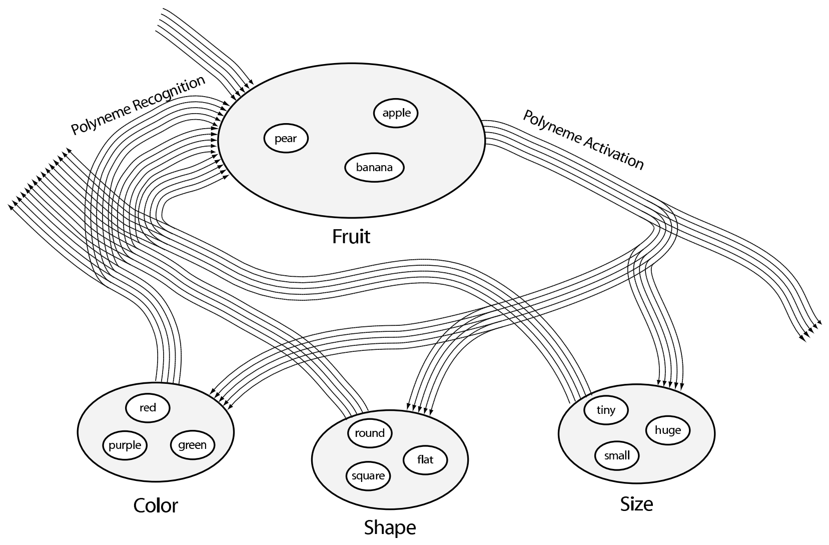


Figure 3.2: Polynemes

Note that under the above scheme, the new recognizer symbol for *apple* is not the same as the *apple* polyneme symbol. One frequent criticism of symbolic AI is that it unreasonably assumes a common, structured symbolic language for representing concepts and communicating them between parts of the brain or program. I am proposing no such thing. Individual agency groups form their own symbol "language", but this language is just an unstructured set of arbitrary labels for long-term memory states. The symbols emerge naturally as patterns that acquire particular significance to agencies that see them.

One way new long-term memories can form is by the creation of new polynemes and polyneme recognizers. This can be done by a scripted process: suppose some learning-management agency decides something important has happened relative to a set of agencies, that warrants learning a new pattern. It can store the agencies' contents in a particular set of short-term memories (Chapter 4) used for learning, then initiate the lengthy process of modifications necessary to create the new long-term memory activators and recognizers necessary to construct the polyneme. This involves allocating some new units that learn the necessary weighting coefficients for connections to the inputs, outputs, and units in the agencies.

Specifically, a new recognizer is allocated in the source agency, which connects randomly to some number of output wires. This generates a new symbol. The target agencies are all trained to build new long-term memory activators associating that symbol with the states stored in their short-term memories. Simultaneously, a new long-term memory activator is allocated in the source agency, and is trained to recognize the combined output pattern that the memories induce in the target agencies.

Of course, this process might disrupt ordinary communication between the agencies involved. Consistent with this observation, there is some evidence that much of long-term memory formation occurs during sleep [20].

3.3 Abstracting Long-Term Memories

Assuming the proposed architecture for long-term memories, simulating networks using them involves simulating a lot of activator and recognizer units, and large numbers of input and output wires mediating the connections.

However, within the constraints of the overall agency topology, connections between units in different agencies may be treated abstractly. That is, units connected by long-term memories may be wired directly together, provided that the source unit's agency has outputs that reach the target units' agency (Figure 3.3). This may be done even while the network is running – that is, new long-term memories may be formed by simply "rewiring" the network.

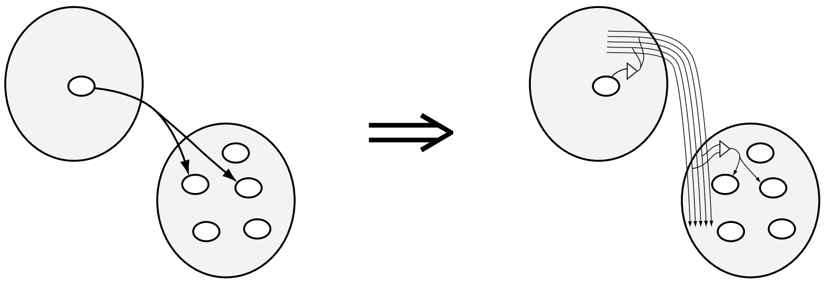


Figure 3.3: Long-Term Memory Abstraction

It is worth emphasizing both the utility of this abstraction mechanism and its limitations.

This technique abstracts plasticity – it lets you connect things up in ways that you can't easily with connectionist networks or behavior-based robotics. Most behavior-based architectures assume a fixed behavior topology, or at least a very restricted mechanism for forming new connections [3, e.g.]. Thus, allowing new connections to form between arbitrary units or groups of units in connected agencies, at run-time, gives a greater degree of flexibility than a typical behavior-based robot has.

However, it gives a *lesser* degree of flexibility than is available in most conventional programming environments. Connections cannot justifiably be made between arbitrary units, only between units in connected agencies. It is not reasonable to assume all agencies are connected, either, because not all regions of the brain are directly connected together. Apart from this biological constraint, allowing agencies to "see too much" of the rest of the network would likely make most kinds of learning much more difficult. Finally, network modifications that are intended to be permanent cannot justifiably be formed instantly; it takes time for long-term memories to form in biological brains. (This last restriction can be relaxed assuming a suitable short-term memory caching mechanism.)

Therefore, the long-term memory models given above may be used to modify network connectivity in a fairly flexible manner, but the kinds of unrestricted pointer assignments possible in ordinary programming languages can lead to programs that dont map well to biological hardware – and thus that, as AI programs, should perhaps be viewed with skepticism.

Chapter 4

Short-Term Memories

In this chapter I present models for short-term memories, which are like long-term memories with the added feature that they may be reassigned very quickly. I describe some useful ways to use short-term memories in unit networks, and I show how to abstract the short-term memory constructions in order to get their benefits without having to simulate their detailed composition.

4.1 Short-Term Memories

It is useful to be able to form memories very rapidly. For example, suppose there is a network that controls a creature which uses some particular agencies to store visual attributes of objects it is seeing. If a red ball, say, is seen, it might be useful to remember the attributes currently stored in some of these agencies, then restore them a short time later – perhaps for use by other processes interested in grabbing the red ball.

Biological brain hardware constraints apparently prohibit the kind of memories I called long-term memories from being used this way, however – it seems to take on the order of half an hour to form a long-term memory. Instead, one needs short-term memories for tasks such as this. They should have the same kind of functionality as long-term memories, plus the ability rapidly to reassign the agency states they activate and recognize.

One way to make a short-term memory for an individual unit is to associate a *mirror* unit with it, which is used to remember its activation level. To reassign the memory's contents, gate the original unit's output to the mirror unit's input. (The mirror unit must maintain its state in order to retain values; one way to do this is with self-input.) To activate the memory, gate the mirror unit to the original unit's input. To recognize the memory (that is, to detect when the unit state matches the state stored in the mirror), use a comparison gate between the original and mirror units (Figure 4.1).

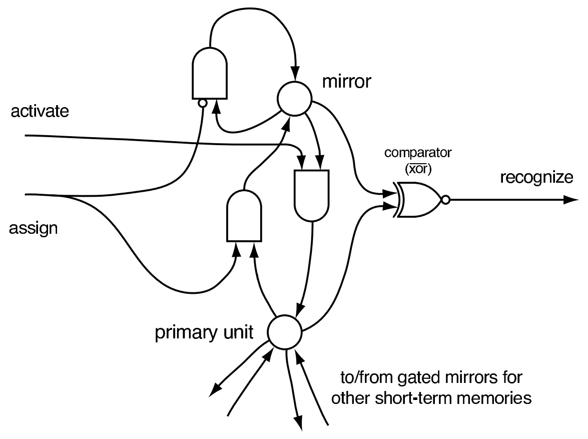

mirror
activate
comparator
($\overline{\text{xor}}$)
recognize
assign
primary unit
to/from gated mirrors for
other short-term memories

Figure 4.1: Short-Term Memory for a Single Unit

To make a short-term memory for an entire agency, use one mirror per unit, and the same kind of pattern-detecting units used for long-term memories to trigger both activation and assignment. To recognize the entire memory, apply a long-term memory recognizer to the collection of comparison gates for the unit/mirror pairs (Figure 4.2).

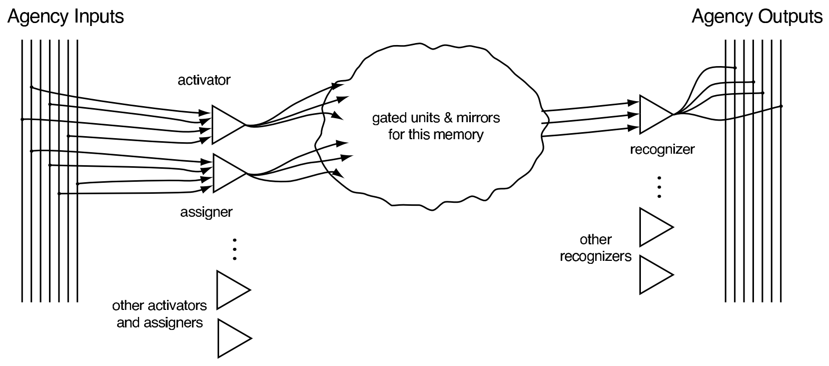


Figure 4.2: Short-Term Memory for an Agency

One can imagine the short-term memories for an agency existing in sheets of mirror units, one sheet per memory (Figure 4.3). This is reasonable biologically – it is easy for genes to build repeated structures, and brains are full of them. Then the connections between source units and mirrors would run perpendicularly through the sheets.

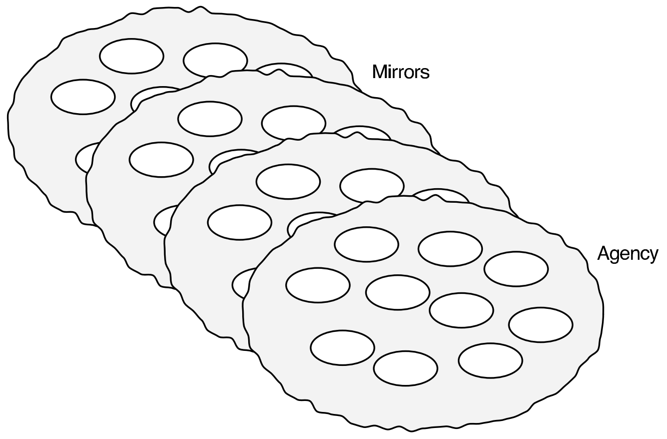


Figure 4.3: Short-Term Memories as Sheets of Mirror Units

There are alternative or hybrid schemes that might be considered for storing short-term memories. The values on the agency input wires could be mirrored, rather than the unit values themselves. This would make short-term memory activation restore previous memories indirectly, by cuing long-term memories, rather than directly. Or some combination of the two techniques could be used. Generally, an agency can have short-term memories for whichever of its units need to be mirrored to support its intended function.

The short-term memory architecture I have described allows arbitrary memories to be compared easily. To compare two memories, first activate one, then store it in a short-term memory, then activate the other one. The degree of match will be reflected by the activity of the recognizer for the short-term memory used.

Interestingly, one feature of the Society of Mind model is the assumption that memories *cannot* be compared in this fashion. Instead, Minsky proposes performing comparisons by noting time changes in response from agencies when quickly swapping in first one memory and then the other [13, Section 23.3].

The ability to do comparisons the way I have described, which turns out to be very useful, depends critically on the assumption that short-term memories have recognizer units. If all the units in an agency have mirrors, then this is easy. However, if instead only the input wires are mirrored (as considered above), it is harder to see how short-term memories can be recognized. But if the output wires are also mirrored, then when the short-term memory is formed a record will also be formed of the agency output, and this can then be matched against current agency output.

One can compare states of many agencies at once, by using short-term memories as polyneme components. Then the polyneme recognizers will detect the simultaneous match of all the target agencies with a particular short-term memory. This possibility is illustrated in detail in the following section.

4.2 Uses for Short-Term Memories

One useful role for short-term memories is as a way for different agencies to communicate. If agency A wants to invoke agency B to perform some task, then the details of that task can be specified by loading the appropriate information into one of B's short-term memories. More generally, two agencies can communicate by using memories of a third agency that is accessible to both.

Suppose there is a set of attribute-representing agencies for *shape*, *size*, *color*, etc., as described in Section 3.2 (Polynemes). Then suppose an eating agency wants to eat an apple. It can activate the *apple* polyneme, putting the attribute agencies in appropriate apple states, then instruct the agencies to store their current contents in a *target* memory (by activating *target*'s assigner), and finally activate a *find* agency which attempts to find whatever is in *target*.

If there is a polyneme for *target*, then *find* can detect when an apple has been found, by checking *target*s polyneme recognizer. More explicitly, when an apple is seen, the *color*, *shape*, etc. agencies will get put into states corresponding to the appropriate apple attributes. Each agency's state will then match its *target* short-term memory, so the individual *target* recognizers will be active. The combined activity of all of them will activate the *target* polyneme recognizer. This works just the way recognizing any polyneme works, but in this case *any* object can be recognized with just one recognizer. This lets that short-term memory act as a common address via which different agencies can communicate.

Note the utility of the short-term memory recognizers. With them, it is easy to construct a simple network that, e.g., automatically finds a given object. Without them, it is not obvious how to perform such a task. One could perhaps imagine that rather than using a particular short-term memory to store *apple*, it could simply be placed directly into the attribute agencies. Their contents would get replaced by attributes coming in reflecting current perceptions, but maybe some kind of "trace" of recent activity would make the agencies more likely to "notice" when they were in the *apple* state again soon after. This is conceivable, but it is not obvious how to associate the detected state with a particular source. That is, the agency might know that it is detecting a recently active state, but that is less information than knowing that it is in the same state as a particular short-term memory.

One more thing short-term memory recognizers can do, when used as part of polyneme recognizers, is detect specific *differences* between memory state and current agency state. This capability is critical to, e.g., Winston's abstraction-learning algorithm [21].

4.3 Abstracting Short-Term Memories

Just as with long-term memories, the constructions I have given for short-term memories could be directly simulated as unit networks. This would require something like five new units per original unit per memory (see Figure 4.1), plus activator, assigner, and recognizer units for each memory.

But again, these constructions may legitimately be abstracted. The simplified version requires three units per memory, total – an activator, an assigner, and a recognizer (Figure 4.4). The assigner is a unit that, when activated, instantly "rewires" the activator and the recognizer so that they directly connect to the currently active units in the memory's source agency.

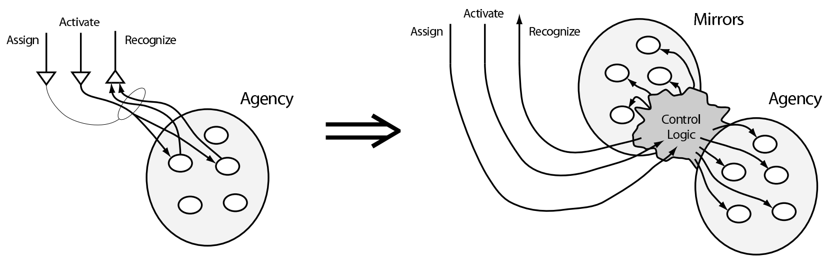


Figure 4.4: Short-Term Memory Abstraction
("Control Logic" represents the constructions in Figs. 4.1 and 4.2.)

It is certainly unusual to think of a connectionist network which allows the connections to be rewired instantly as it is running, but such a thing is perfectly justified by treating the units that are rewired as denoting the explicit short-term memory structures given above.

Again, such operations are subject to agency topology constraints. Furthermore, the number of short-term memories allocated to a given agency should not be too large. What is too large? I dont know, but on the order of a dozen doesnt seem unreasonable. One should perhaps be wary of networks that need hundreds or thousands of short-term memories assigned to any one agency.

Likewise, programs that need large memories that can be indexed as arrays don't seem to be translatable to reasonable networks – even if the quantity of memory were available, there would have to be some digital representation of the index.

Chapter 5

Frames

In this chapter I develop the important AI idea of *frame*, and describe how to build frames in unit networks. After showing that there are some problems with the simplest approach, I propose an improvement to the short-term memory architecture described in Chapter 4, and show how this makes frames much more tractable and powerful.

5.1 The Frame Idea

The concept of *frame* [11] has been an important element of much AI research. Briefly, a frame is a structure that represents a particular kind of object or situation, where the details are to be filled in by specifying values for *slots*. For example, a frame for representing the contents of a room would have slots for walls, furniture, carpet, etc. A frame for representing a person could have slots for particular parts such as head, arms, and legs; it could also have slots for attributes in different domains – say social attributes, such as gender, age, and relationship. Frames can also represent sentences or larger linguistic structures (Figure 5.1).

Frames' utility lies in their enabling some kind of action or reasoning based on slot contents. For example, a room-frame could be used when performing an action such as opening a window, by using the attributes of whatever was currently attached to a *window* slot of the frame. A person-frame would be useful in a conversation, by using the information in the *relationship* slot to modify goals and expectations of the conversation. A "trans-frame" is a kind of frame representing general change of state from one condition to another. Trans-frames can

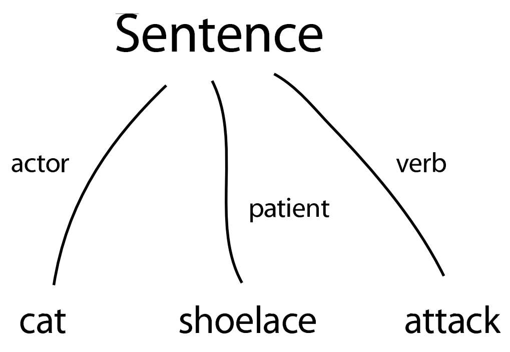


Figure 5.1: A Sentence-Frame

facilitate a certain kind of reasoning, by chaining one frames *destination* slot to the next ones *source* slot.

The true power of frames comes when they are used in *frame systems*, which are ways of using frames to select new frames using the same slots, to shuffle slot assignments, or to transform slot contents.

For example, a frame might represent a particular view of an environment. Information that has been accumulated about the environment can be partially represented as slot contents. When the view changes slightly, due to the motion of the observer, the view contents will change in a way that is largely predictable. This can be captured by using a frame system. The currently active "picture-frame" detects the motion, and can select a new frame to better represent the new situation, or reassign some of the slots, or modify the contents of some of the slots. This is much more effective than completely regenerating all the information about the current environment on a moment by moment basis.

Finally, frames can represent *defaults* in terms of built-in slot assignments that can be overridden. For example, a generic office-frame might have slots for *desk*, *chair*, etc. initially filled in with a default chair-frame, desk-frame, etc. But the frame can be applied in a specific situation, by overriding the defaults with specific slot assignments as they become available. The existence of defaults lets one reason effectively using typical aspects of situations when specific aspects are unknown.

5.2 Frames in Terms of Units

To use frames in the kinds of unit networks under discussion, they must be represented in terms of some kind of unit structure.

A natural way to consider representing frames is by analogy with a conventional "record" structure. A frame can be a unit which effectively has selectable slots, by making it activate half the inputs of a set of AND-gates, the other half of whose inputs are activated by some sort of field-selector units (Figure 5.2). So activating a particular sentence-frame and the actor selector symbol at the same time will cause whatever memory is in the actor slot to become active. These other memories can be polynemes, frames, or any sort of unit.

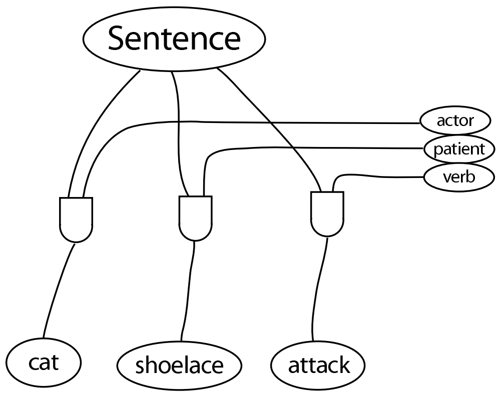


Figure 5.2: Frame as Units: Slot Selection with AND-Gates

5.3 Slot / Short-Term Memory Relationship

Since a frame must be able to represent a variety of situations, depending on slot contents, it makes sense that the slot contents have to reside in short-term memories. However, given this, it would seem that the field-selectors described above have no purpose. For suppose an agency wants to activate the *actor* slot of an active frame. It activates a selector unit, which in conjunction with the frame unit activates an AND-gate that does – what? What needs to be done to activate the *actor* slot is simply to activate the *actor* short-term memory, since that is the slot where the actor memory is stored. It would seem that the frame unit itself, and the AND-gate, have nothing to do with this. The *actor* field-selector unit can simply be the *actor* short-term memory activator unit. (All short term memories have activator, recognizer, and assigner units – see Figure 4.4.)

However, this analysis disregards the default slot assignments. Those are stored in long-term memories, and must be selected based on the frame as described. That is the purpose of the AND-gates.

When a slot is selected, both the default assignment and the values in slot's corresponding memory are activated at the same time. Minsky's idea is that the default is activated weakly, and any conflict resulting (for example, involving cross-exclusion links) will be resolved in favor of the actual slot assignment.

(One interesting question about using short-term memories as frame slots is whether memories can be re-used for different slots of different frames. One can imagine a wide variety of kinds of frame, with even more kinds of slots. If agencies had to reserve an entire short term memory (which is after all a copy of the agency) for each potential slot, that might result in a lot of wasted space. Maybe there are only a small number of actual short-term memories for a typical agency, which can be re-used to get any number of slots – as long as no one frame, or common combination of frames, needs more slots than there are short-term memories.)

5.4 Specialization

As detailed knowledge is acquired filling in the slots for a given frame, it can be useful to remember that knowledge by somehow storing those slot assignments in long-term memory. For example, a generic person-frame should gradually be refined into a specific person-frame for representing information about a particular friend. The obvious way to do this is with the defaulting mechanism. A new frame unit can be formed, as a new long-term memory, with some of the current slot assignments turned into default assignments of the new frame. A default can even be strengthened into a constraint, by increasing its activation strength. Once formed, the new frame can continue to refine its default or required slot assignments by further long-term memory modifications.

This makes slots seem a little bit schizophrenic, with some of the values residing in short-term memories, and others in long-term memories accessed via slot selectors. This could cause problems when one frame acts as part of a frame system to transfer control to a different frame using the same slots. Since some or all of the slot contents were not stored in the short-term memories, those contents would be lost.

For example, a generic person-frame might have links to switch to an "antagonist-frame" when a conversation starts to turn into an argument. If the specialized friend-frame switches to this frame, then the information stored in long-term memory about that friend will not be accessible in the new context.

5.5 Sequential Activation, Complex Recognition

The problem raised in the last section, and similar issues, mean that frames need to be completely "unpacked" into short-term memories before being effectively used. This involves activating each slot selector in turn, and then storing the active agency contents into the appropriate short-term memories. Then other frames or behaviors needing the slots can have ready access to them.

This is a rather inconvenient process to have to go through to activate a frame, especially when this unpacking process must be controlled by the unit network itself.

However, *recognizing* frames – that is, determining from current context when a given frame should be active – is even harder. Consider the friend-frame described above. When you see that friend on the street, how do you recognize him or her? The knowledge stored in the details of the slot assignments should be available for this kind of recognition. However, the frame unit doesn't work the way a polyneme does – it can't have a simple recognizer that detects simultaneous activation of all the component agency recognizers. This is because the information is stored in many different slots. Something like the reverse of the unpacking process needed for activation must happen, but it is more complicated. As you observe the person on the street and note various attributes, your current mental model is accumulating detail as a set of short-term memory assignments. But to match those against a frame, somehow the frame has to detect the total sequence of "this slot, that assignment", "that slot, that other assignment", etc., and match the assignments against its own slot values. But the sequence is not guaranteed to occur in any particular order, since you might observe various aspects of the person at various moments.

This is is one aspect of the "frame recognition problem", which is the general problem of how to select the best frame to use for a given situation. Various schemes to handle it have been proposed, all of them complex [11].

A simpler illustration of the frame recognition problem is that a chair cannot be recognized simply by the simultaneous detection of all of its parts. The relationship between the parts (that is, the roles or slots those parts occupy) is also necessary information.

5.6 Frames as Super-Polynemes

It is possible to simplify the frame problems quite a bit, by adding more power to the short-term memory model given above. Specifically, it would be very useful if short-term memories, which are essentially copies of their source agencies, could somehow form and recognize their own long-term memories in the same ways that the source agencies that contain them do.

For purposes of frame activation, the problem is that the default slot assignments must be loaded into the short-term memory slots. If the short-term memories could simply learn to associate a given frame symbol with the appropriate default assignment, this problem would be solved.

Consider the analogous situation for an ordinary polyneme, such as *apple*. All the agencies that receive the *apple* symbol have learned to associate it with their own particular internal states. This is done by allocating long-term memory activator units and adjusting their weights (Section 3.1). Since short-term memories are really copies of entire agencies, its conceivable they could do the same thing.

For example, imagine a child learning how to throw a baseball for the first time. He already has some experience with throwing toy balls, so he might have a generic frame to use for throwing balls. Suppose that a new frame is formed, to become a specialized version of the throwing frame, just for baseball. When throwing a baseball, the contents of the *ball* slot are always the same – *hard*, *white*, and *apple-sized*. The agencies that represent those attributes – *feel*, *color*, and *size* – each have a short-term memory that's used for storing *ball* attributes, for whatever kind of ball is currently being used. If those short-term memories work just like the agencies they are copies of, then they can learn to associate the new baseball-throwing frame with their appropriate states for a baseball.

This would solve the activation unpacking problem. But if short-term memories can form internal long-term memory activators, then they should be able to form internal long-term memory recognizers as well. That means they can put particular symbols on the output wires when they are in particular states, just as the agencies themselves do.

And that means that *frame recognition becomes easy* under this model. Frames can be recognized in exactly the same way as polynemes, by the simultaneous activation of the recognizers for the relevant attribute agencies (Figure 3.2).

The implementation concept of a frame here is completely different from that described in Section 5.2 – there are no AND-gates with which to select slots. When a frame is active, *all* its slots are effectively activated at once, because it's just like a polyneme (Figure 5.3). All of them engage in the normal default-resolution process that previously required individual slot selection. There is no need to unpack the frame sequentially.

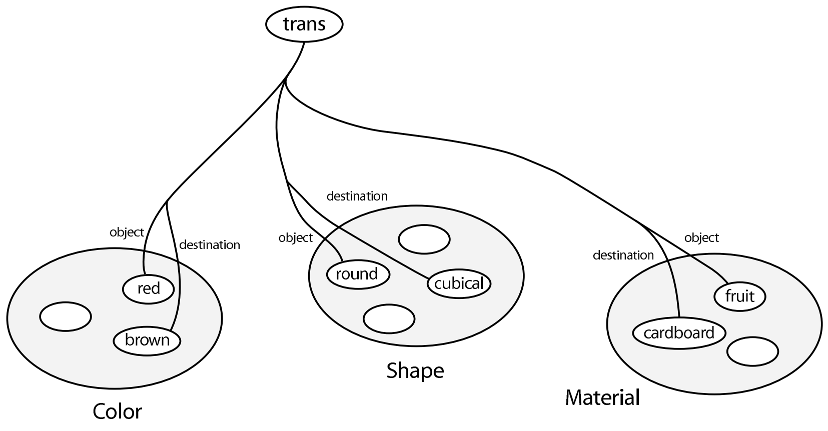


Figure 5.3: Trans-Frame as Super-Polyneme

I call frames under this scheme *super-polynemes*, because they act just like regular polynemes, but can directly address short-term memories as well as agencies.

Let's go back to the problem of recognizing the appropriate person-frame based on seeing a friend on the street. Once enough visual detail has been stored into the relevant short-term memories, those memories will collectively activate the right person-frame – regardless of the order in which they are filled in. This works because the person-frame can receive output symbols from those short-term memories, and learn to recognize the correct combination of them.

5.7 Frame Abstractions

No new kinds of constructions are needed to support the abstraction of frames as super-polynemes – simply assume that the short-term memory mirrors (Figure 4.3) also have the same internal construction as their agencies (Figure 3.1).

This assumption does allow for a new kind of unit relationship, however. When a unit acts as a symbol for a short-term memory rather than an agency, that relationship deserves to be labeled. Wires that load particular long-term memories into short-term memory states should be represented as connected to those long-term memory states, with labels indicating the short-term memory. This is illustrated in Figure 5.3, which shows a part of a trans-frame for "put the apple in the box".

The only apparent problem with this scheme is that it might seem to require that symbols be learned multiple times, one for each role they can occupy. For example, an agency that loads an *apple* state in response to a particular symbol might have a short-term memory that loads an *apple* state in response to a completely different symbol.

However, this problem is illusory, because it is not really *apple* that the short-term memory has learned – it's a specific frame, which simply happens to want *apple* in that short-term memory slot. The symbol for *apple* only has to be learned once. After that, the corresponding state can be loaded into any short-term memory, and thus occupy any role.

This abstraction mechanism is very powerful, and it blurs the line between frames and other units. It means any unit can effectively activate or recognize units not just in other agencies, but in particular short-term memories of other agencies.

For example, the *apple* polyneme could be taken to have some frame-like properties as well. Specifically, apples have insides and outsides, with different properties. Those could be frame slots. Thinking of an apple's surface will activate one set of attributes, and thinking of its interior will activate different ones.

5.8 Hierarchical Frame Representations

The previous example might seem to raise a serious problem: if the *apple* symbol is being used as an item in a frame slot, and it has frame slots itself, then wont the slots interfere when the parent frame is active? How can the apple attributes be loaded into slots of attribute-representing agencies, when rather than constituting specific agency states, the apple attributes are themselves distributed over multiple short-term memories? In some sense, *apple* seems to be "too big" an object to load into a single slot.

This situation is really just a particular case of the general issue of representing hierarchical frame structures in short-term memories. To do this, agencies at different levels must be involved. At some level, there is an agency that contains a unit (or set of units) which represents the *apple* frame. When that unit is active, it puts the lower-level agencies' short-term memory slots into appropriate states: the *inside* slots are loaded with attributes for the insides of an apple, etc.

The agency containing the *apple* frame unit itself is the one whose short-term memories can meaningfully hold *apple*. The trans-frame must use slots of that agency to represent objects which are frames using slots of lower-level agencies.

It might seem that frames are thus restricted in the kinds of objects that can occupy their slots, since they must target particular agencies. But actually, units of any agency can activate frames and other units in arbitrary other agencies (subject to overall agency topology), so one can use a kind of indirection to link different kinds of frame structures together (Figure 5.4).

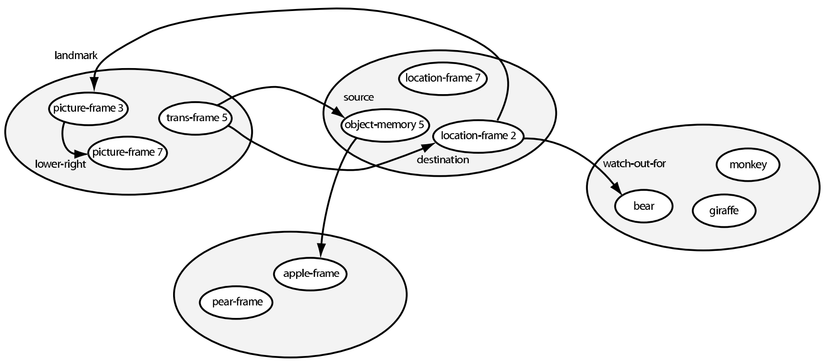


Figure 5.4: Hierarchical Frame Structures in Short-Term Memory Slots

I use this approach for implementing trans-frames in my simulation. There, the high-level slots hold scripts which can perform arbitrary memory operations. The scripts used load low-level attributes into the slots that are needed by other behaviors the trans-frame invokes.

Of course, there is only so much short-term memory available in a network, and therefore it can only represent frame structures up to a certain size. But this is a limitation people have, as well. When a problem becomes too large to solve "in your head", you resort to making notes and diagrams – a form of external memory.

One last remark I should make is that Figure 5.4 seems to resemble a general semantic net, so the question might arise of why I dont just build semantic nets in LISP to begin with.

The answer is that typically operations are performed *on* semantic nets, but I am interested in what happens *in* unit networks. The difference is that arbitrarily deep recursive procedures are not permitted *in* finite networks.

Chapter 6

Applicability of Results to Society of Mind Model

As I remarked in the introduction, Marvin Minsky's "Society of Mind" (SoM) model inspired much of this work.

This chapter serves both as a dictionary mapping SoM concepts to concepts in my model, and as a summary of the contributions and extensions I feel I have made to the SoM model.

6.1 Agents

I have avoided using the term *agent*, because since SoM was published the word has come to have too many different meanings. Even within SoM, *agent* sometimes means what I call *unit*, and other times corresponds more to collections of units or entire agencies. The term is used in a functional sense in SoM, so the issue of actual representation is secondary. This work focuses on representation, so I refer instead to units, assemblies of units, or agencies, depending on the particular actual structure I am discussing.

6.2 Agencies

I use the term *agency* in a very specific sense, to describe a collection of units with particular kinds of memory interfaces. In SoM *agency* often refers more to functional characteristics of agents or groups of agent rather than structural or representational characteristics.

In fact, my usage is probably overly specific, in that the boundaries between agencies probably need to be more fuzzy than they are in my model (see Level-Bands, below).

6.3 K-lines

K-lines [12] are the mechanism of memory formation in SoM. A K-line is formed by attaching to recently active agents in a situation where some useful problem has been solved, or some other situation has occurred that warrants remembering the current state.

My long-term memories are generally in correspondence with permanent K-lines, and my short-term memories provide an implementation of temporary K-lines – devices that were mentioned, but not described in much detail, in SoM [13, Sec. 22.1].

One aspect of short-term memories that I consider a valuable contribution to the SoM model is that they have recognizers, just as long-term memories do (Section 4.2). This lets arbitrary memories be compared easily, without the time-blinking proposed in SoM [13, Sec. 23.3]. Since I have actually done behavioral programming in an SoM-inspired framework (Chapter 7), I can report that this capability makes many behaviors much simpler to construct than they would otherwise be.

Short-term memories, as copies of agencies, are also permitted to form their own long-term memories in my model; this solves several problems with frames.

There are two important things missing from my model relative to a full implementation of K-lines, however. The first is a learning mechanism. I have sketched the means, but not the control process, for forming new long-term memories.

The second thing missing is level-bands, below.

6.4 Level-Bands

Level-bands are one of the theoretically most powerful, but least well specified, aspects of K-line theory. When new K-lines are made, they connect most strongly at a particular "level of abstraction" some distance below the current level of activity, in some sense. Various kinds of interactions at the "fringes" of the level band can have profound computational effects.

I have not yet implemented any aspects of level bands, but I am trying to extend my model in order to do so.

6.5 Nemes

A *Neme* is an agent whose output represents a fragment of an idea or a state of mind. It is more or less what I call a symbol, which is an agency input/output pattern corresponding to a particular long-term memory.

6.6 Polynemes

I use the term *polyneme* generally as Minsky does. Polynemes are described in Section 3.2. One apparent addition to the SoM model is the ability to treat short-term memories as full-fledged polynemes, with accompanying recognizers (Section 4.1). Furthermore, I have provided what I think is a more detailed description of how polynemes could be learned (Section 3.2) than is found in SoM. Finally, I have extended the polyneme concept in that I treat frames as "super-polynemes" (Section 5.6).

6.7 Micronemes

Nemes are represented by combinations of low-level agents called *micronemes*. They correspond roughly to agency input/output wires, but in my system they do not have an individual meaning, as they do in SoM [13, Sec. 20.5]. In fact, they are below the abstraction level in my system, and in my implementation.

6.8 Frames

Frames [11] as used in SoM are described in Sections 5.1 and 5.2. The biggest overall difference in my model from the SoM model involves the structure of frames. I describe some problems with building frames in terms of unit networks in Section 5.5, and in section 5.6, I argue that some additional capabilities of short-term memories solve these problems. The frame structure I propose does not involve component selection via AND gates; instead, *all* the slot are active at once; they are loaded directly into appropriate short-term memory slots.

Frames can be viewed as a generalized kind of polyneme under this model.

6.9 Pronomes

The term *pronome* is used in SoM to refer both to frame slot selectors and to their associated short-term memories. In my system, the selectors are synonymous with the short-term memory activators, and I generally use the term *slot* to describe a short-term memory when viewed as a component of a frame.

Chapter 7

Implementation

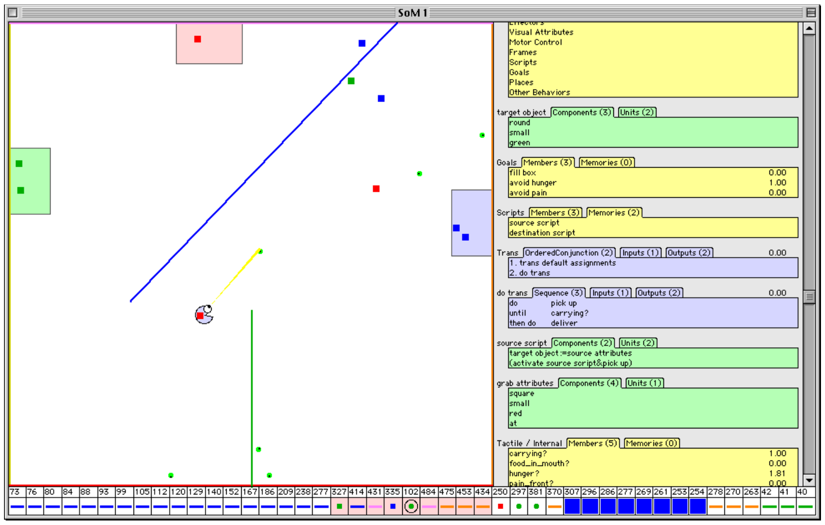
I have implemented many of the above ideas in a C++ program that simulates a creature in a 2D world. Its environment consists of walls, boxes, blocks, and food. Its primary task in life is to pick up blocks, and deposit them in boxes with matching colors. It also gets hungry, and needs to eat food. The food can be thought of as prey – it moves around, and must be chased down and caught. The creature also registers pain when it runs into walls.

The creature's interface to the world is via a biologically plausible set of sensors and actuators. It has no absolute location knowledge or direct knowledge of world state, for example. Given this world interface and the unit-based creature design, various kinds of short-term memory control processes and frames are needed to perform the desired behaviors.

A sample screen is shown in Figure 7.1. The configurable display on the right shows memory contents and unit activation values. Behaviors may be closely monitored and debugged. For example, the fact that the creature is carrying a red block is reflected in the contents of the *grab attributes* short-term memory of the visual attribute agencies. The display across the bottom contains the contents of the retina. Only the attributes from the target area in the fovea are directly available to the creatures sensory network.

7.1 Overall Architecture

The creature control mechanisms are implemented as a network of approximately 350 interconnected computational units, similar to those described in Chapters 2–5. It might seem that an organism with 350



```
Visual Attributes
Motor Control
Frames
Scripts
Goals
Places
Other Behaviors

target object  Components (3)  Units (2)
  round
  small
  green

Goals  Members (3)  Memories (0)
  fill box                                    0.00
  avoid hunger                                1.00
  avoid pain                                  0.00

Scripts  Members (3)  Memories (2)
  source script
  destination script

Trans  OrderedConjunction (2)  Inputs (1)  Outputs (2)   0.00
  1. trans default assignments
  2. do trans

do trans  Sequence (3)  Inputs (1)  Outputs (2)          0.00
  do        pick up
  until     carrying?
  then do   deliver

source script  Components (2)  Units (2)
  target object:=source attributes
  (activate source script&pick up)

grab attributes  Components (4)  Units (1)
  square
  small
  red
  at

Tactile / Internal  Members (5)  Memories (0)
  carrying?                                   1.00
  food_in_mouth?                              0.00
  hunger?                                     1.81
  pain_front?                                 0.00
```

Figure 7.1: A Hungry Creature Hunts Some Prey.

"brain cells" would not be capable of any very sophisticated behavior, but each unit represents many more primitive units. The connections between units in different agencies represent spatial patterns on large bundles of wires (Figure 2.4). Each of the short-term memories represents an independent copy of the units in its source agency, along with control logic (Figure 4.2).

The network is run as a discrete-time simulation. I experimented with various update strategies, including synchronous updating and stochastic updating. The network behavior proved relatively insensitive to updating strategy, so I settled on a simple fixed-order sequential update. Units recompute a floating-point activation level when updated, and use either a threshold function or a simple sum pinned below at 0.

At each time step, all the units are recomputed, then the actuators modify the creature's physical state, then the world objects update their states, and finally the sensor units are updated with new sensory input. There is no process of "settling the network" between world interactions, thus the creature's brain is forced to propagate information fast enough to keep up with the world. Relative to simulation time, units are updated approximately every 5 msec. – in the same ballpark as neuron propagation delay times. However, this comparison is not very precise, because units don't map directly to neurons. Nonetheless, it is a good idea to design behavior architectures that don't require thousands of propagations to react.

7.2 Sensors

Vision. The creature possesses a one-dimensional retina, 45 cells wide – essentially a z-buffer image of its current field of view. The retina is part of an early vision system which is outside the unit simulation proper. This system mediates between the world simulation and the units in the vision agencies. It maintains a "focus point", which corresponds to a spot on the retina with motion tracking. When the focus point is in the fovea, the early vision system reports quantized values of distance, color, shape, and size to units in the vision agencies. It remembers how long it has been since each part of the visual field has been focused on, and from this computes integrated *left interest*, *right interest*, and *ahead interest* values. It also registers the degree to which the eye is turned left or right, and reports when the focus point has drifted to the left or to the right of the the center of the retina. Finally, it reports on dangerously close large objects (walls) to the left, to the right, or ahead.

On the output side, the early visual system responds to an actuator unit which resets the focus point to the center of the fovea.

Though the retina represents a wide (90 degree) field of view, the creature (i.e. the unit network) can only directly perceive one set of visual attributes at a time. Information on the rest of the retina is reduced to visual interest values and proximity detectors. To build up a more complete picture of its current surroundings, the creature must foveate to different parts of the visual field, accumulating previously perceived relevant information into short-term memories, or otherwise modifying current state.

Proprioceptive Sensors. The creature has sensor units detecting whether it is moving forward or backward, turning left or right, or opening or closing its mouth.

Tactile / Internal Sensors. There are sensor units registering pain in the front or the back (resulting from collisions with walls). There are also units detecting food in the mouth, an object being carried, and hunger level. Hunger increases with time and decreases when food is eaten.

7.3 Actuators

Body Actuators. There are actuator units which make the creature move forward, move backward, turn left, turn right, open or close its mouth, pick up an object at its current location, and drop any object it is carrying.

Eye Actuators. The eye actuator units turn the eye left and right, and (as previously mentioned) reset the focus point to the center of the fovea.

7.4 Behaviors

Some of the behaviors described here are diagrammed explicitly in Appendix A.

7.4.1 Object Location

Most creature behaviors involve locating objects of particular types. This is done by loading the desired attributes into a *target object* short-term memory of the visual attribute agencies. For example, food is represented in the system as small, round, green objects. In order to locate food, the creature loads these attributes into the target object short-term memories of the size, shape, and color agencies (by activating a *food* polyneme directed to those memories).

Low-level vision behaviors cause the eye to foveate from place to place until an object with the desired characteristics is located. This detection is manifested by the recognizer for *target object* polyneme becoming active, indicating that the attribute agencies are all in a state matching that memory.

Low-level motion control behaviors cause the creature to turn left or right whenever the eye is respectively left or right. When combined with the ability to track the focus object, these behaviors cause the creature to find and point itself toward a matching object – when one is in the field of view. When there is no such object in the field of view, the eye will be turned maximally left or right trying to reach the unexplored portion of the visual field, thus causing the creature to rotate around and explore its complete immediate environment.

Together, these behaviors constitute an implementation of the "automatic finding machine" Minsky describes [13, Sec. 21.5].

Generally, a *go to it* behavior will be running, since the creature is usually searching for some object or other. This causes the creature to move forward when *target object* is seen. In combination with the object location behaviors, this makes the creature find and go to an object or location matching *target object*. This works even when the desired object is moving – e.g., food, which wanders around randomly.

7.4.2 Picture-Frames

Since the creature can only directly perceive one set of attributes at any given instant, it must build up a memory-based model of its environment. This is done by storing "landmark" information (wall and box locations) in short-term memories representing the landmarks to the left, to the right, and directly ahead. As described in Chapter 4, these short-term memories constitute the slots of picture-frames. The main frame system using these slots shuffles the slot assignments as necessary when the creature turns. For example, when the creature turns sufficiently far to the left the *ahead-landmark* becomes the new *right-landmark*, the *left-landmark* becomes the new *ahead-landmark*, and the currently perceived landmark, if any, becomes the new *left-landmark*.

At any given moment, the landmark memory matching the eye direction gets updated with currently seen landmark attributes. This process is a simple implementation of direction-neme-controlled temporary K-line selection, as described in [13, Sec. 24.8]. A more sophisticated implementation would build up hierarchical frame structures into a picture-frame.

Various picture-frame instances recognize particular landmark configurations, and activate units in a location-representing agency. This is an example of the utility of the concept of frames as super-polynemes (Section 5.6). In a system without this kind of model, recognition would have to take the form of a complex process of perception-sequence pattern-matching. As mentioned, frame recognition is traditionally viewed as a hard problem – this is an instance of my model reducing the task to what is essentially polyneme recognition. Of course, the picture-frames I have implemented so far are particularly simple. But the same technique can be used to recognize more complex frame structures as well.

7.4.3 Navigation

The location-representing agency maintained by the picture-frames, when combined with knowledge of location relationships, lets the creature navigate from place to place. For example, if it is looking for the red box, but it is in a location from which that box can't be seen, then it can pick a series of landmarks to find that will get it there. This is done incrementally, by loading the *target landmark* memory with whichever landmark is appropriate in the given location, given the goal location (as represented by the *target location* place memory). The object location behavior will lock onto the target landmark when the target object itself is not present. This is sufficient for the *go to it* behavior to follow the landmark trail until the goal is in sight.

In practice, this means that if a red block, for example, is picked up in a location where there are walls between the creature and the red box, it will successfully follow a path around the walls to the box.

In this implementation, the location map is encoded explicitly rather than learned. The "explicit encoding" is nevertheless in terms of a distributed representation within the unit network, and not in some external conventional data structure.

7.4.4 Trans-Frames

The creature has a general capability, implemented as a "trans-frame", to take a given object to a given location. As in the case of picture-frames, the trans-frame slots are realized as short-term memories. In this case there are *source* and *destination* slots which are short-term memories of a memory control script-representing agency. The trans-frame uses these to find its source and destination. First it activates *source*, which causes some script to be activated. This should be a script which loads the *target object* short-term memory with the desired source object attributes. The trans-frame then invokes a *get it* behavior, which goes to the target object and picks it up. (*Get it* may also be thought of as a frame, since it is a behavior based on a slot (short-term memory) assignment.) After the object has been picked up, the trans-frame activates the *destination* script (which loads the destination attributes into the target object memory), runs a *go to it* behavior, then drops the object.

This is the most general form of trans-frame behavior, but the frame also implements Minsky's concept of default slot assignments [13, Sec. 8.5]. In particular, generally a full-blown script is not necessary to determine what the target objects should be. If the source and destination objects are known in advance, then simpler visual attribute slots may be used instead of script slots. Then the trans-frame will invoke default *source* and *destination* scripts, which will in turn load the attribute slots into the target object memory. For example, to carry some food to a wall, an agency should first load the food attributes into a *source object* short-term memory of the attribute representing agencies, load the wall attributes into a *destination object* memory, then activate the trans-frame.

If instead one set of attributes is not known in advance (see Block-Sorting, below), then the default behavior may be overridden, by putting a controlling script in one of the script slots. The default scripts are activated weakly by the trans-frame, and cross-exclusionary links let explicit slot assignments override the default behavior smoothly. This is exactly the way Minsky proposed that defaults could be used with K-lines [12].

However, it should also be noted that in this implementation frames are not quite so distinguished from other kinds of units as they tend to be in the SoM picture. Again, this is because my short-term memory model lets me treat slot contents uniformly with other units. Frames are just units which can directly activate or recognize long-term memory patterns in short-term memory slots as well as in their source agencies.

7.4.5 Block-Sorting

As noted above, the default trans-frame behavior is not sufficient when either the source or the destination attributes are not known in advance. One example of this is the creature's block-sorting task. It has a high-level goal of picking up blocks and depositing them in boxes which match the blocks' colors. So if a blue block is picked up, then it must be deposited in the blue box. Therefore, this behavior cannot use the trans-frame's default behavior – the destination attributes are not completely known until after the block has been picked up, but the trans frame had to already be running in order to get the block in the first place.

In this case, the block-sorting task overrides the default trans-frame behavior by loading an appropriate script into the destination script slot. This script, which is run after the source object has been picked up, loads the target object attributes with the correct values, partially determined by the attributes of the object that was just picked up.

The creature cannot actually perceive the attributes of the object it is carrying, so this last operation requires the use of another memory. The behavior used to pick up objects also stores the attributes of what is about to be picked up in a *grab attributes* memory, which is then used by the block-sorting destination script.

7.4.6 Hunger Avoidance

Avoiding hunger is another high-level goal of the creature. The high-level goals are those behaviors which tend to require dedication to a particular task. Block-sorting, hunger, and pain avoidance fall into this category, and all are members of a cross-exclusionary agency, to ensure that only one is trying to control the creature at any given time. Other behaviors, such as object avoidance during navigation, can function concurrently with high-level goals – they do not need to be cross-excluded.

When the hunger level has increased sufficiently, the input to hunger avoidance is sufficiently strong to counter the cross-exclusionary inhibition from block-sorting, and eating behavior takes over. This involves loading the food attributes into the target object memory, activating *go to it* (also used by *get it*, above), then executing the appropriate actions (opening and closing the mouth at the right time) when food is reached.

7.4.7 Pain Avoidance

Pain avoidance is another high-level goal. When pain is sensed (by running into a wall), control is immediately wrested from whichever other goal is running at the time, and the creature moves (either forward or backward) so as to eliminate the pain. To keep the previous goal from immediately reactivating when the pain is gone, a *hurt* unit is loaded with a value which gradually decreases – keeping the pain avoidance behavior running while there is still a memory of recent pain.

The creature tries to avoid running into walls in the first place, but occasionally cannot help it, since it might be focused on some object and not notice the approaching wall in time to stop.

7.4.8 Boredom

When the creature is not moving forward, a boredom accumulator increases. When it reaches a critical point, the creature assumes that whatever behavior it is trying to perform is not successful, and tries a change of venue. It does this by picking a location far from its current one, and placing this in the *target location* short-term memory that is also used in ordinary navigation.

In practice, this behavior becomes active when the creature is looking for food or a block in a location where there is none. Wandering to the other side of the world generally fixes this problem.

7.5 User Interface

The simulation is implemented as Macintosh application, with an interface window (Figure 7.1) and menus selecting various control and display parameters. The world as viewed from above is displayed in the main part of the window. The retina contents are represented along the bottom. On the right is a configurable scrolling panel which allows inspection of agency, memory, and unit values.

The visual update rate may be adjusted, and the simulation may be stopped and single-stepped. When stopped, the creature may be driven manually in order to position it in test contexts.

The unit inspection panel has proven to be an invaluable aid in inspecting and correcting creature behaviors – essentially it serves as a unit network debugger. The relevant values may be selectively displayed and observed as the simulation is single-stepped.

Chapter 8

Contributions

I have defined a simple, biologically plausible computation model which can be used as a “ground level” for building up useful AI programming abstractions.

I have defined abstractions for long-term memories, which allow networks of simple units to be wired up flexibly. These abstractions are justified by the mechanism of Zatocoding [16].

I have defined abstractions for short-term memories, which allow a form of controlled instant network rewiring. They embody a realization of the “temporary K-line” concept. Beyond this, they make it easy to compare arbitrary memories directly. The abstractions are justified by specific constructions in terms of logic gates.

I have proposed a novel architecture for building frames out of gate-like units. The key feature is the ability to directly address short-term memory contents. This architecture makes it easy to recognize frames implemented as networks of units.

I have demonstrated applications of all these abstractions by programming a simulation of a creature controlled by an abstract unit network. The creature can navigate around its world, hunt prey, and sort colored blocks into matching boxes. It interacts with a dynamic environment via a biologically plausible set of sensors and actuators, yet is driven by symbolic processes that use memories, scripts, picture-frames, and trans-frames.

Appendix A

Selected Creature Behavior Diagrams

In this appendix I give diagrams for some creature behaviors.

Agencies are represented as large shaded ovals. Within agencies, ovals are long-term memories, and rectangles are short-term memories. Input wires to memories activate them; output wires from memories recognize them (using abstractions in Sections 3.3 and 4.3).

As discussed in Section 5.7, wires that load particular long-term memories directly into short-term memory states are represented as connected to those long-term memory states, with labels indicating the short-term memory. In some diagrams labeled wires connect to short-term memories as well; in this case the meaning is that the contents of the target short-term memory are loaded into the labeled short-term memory. (I have not provided an explicit mechanism for this detail; it can be interpreted as first activating the target memory, then assigning the labeled memory.)

Labeled wires connecting to an agency instead of to a memory within an agency load the current agency state into the labeled short-term memory (that is, activate the labeled short-term memory's assigner unit).

Labeled *output* wires from a long-term memory represent recognition of that long-term memory being stored in the labeled short-term memory.

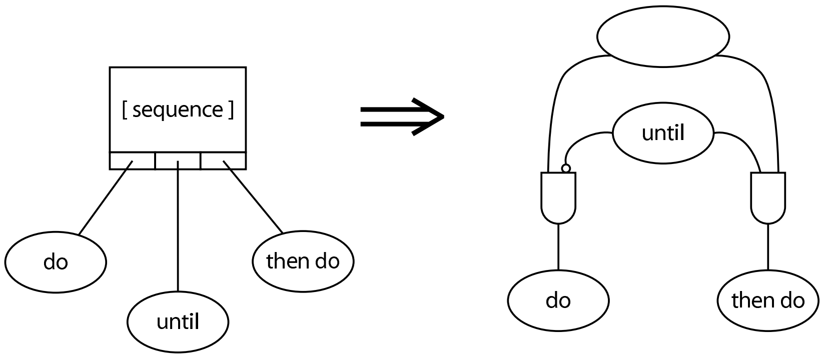


Figure A.1: Sequence Abstraction

When the sequence is activated, *do* will be activated until *until* becomes active; then *then do* will be activated. This is logically equivalent to an *if-then-else*, but the sequence construction is typically used to perform an action until a condition becomes true, then perform a different action.

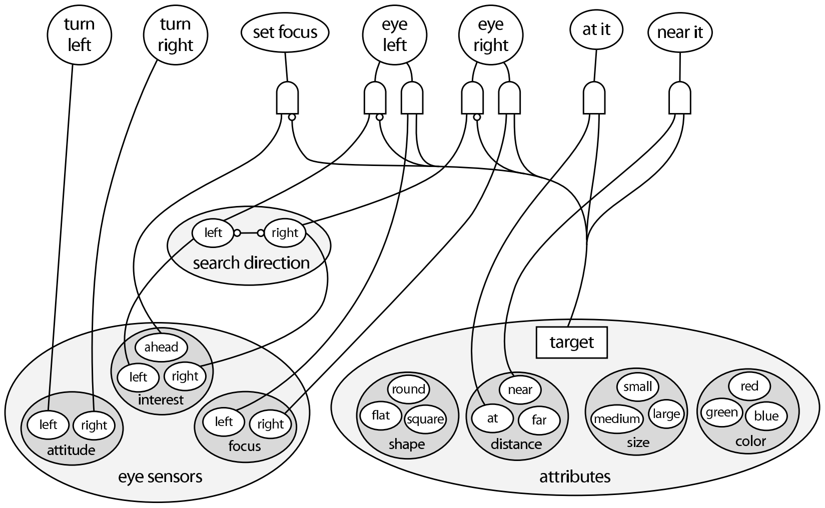


Figure A.2: Vision System (Simplified)

Eye attitude (left or right) controls body turning. When *target* is seen, the focus spot is tracked; otherwise, the eye moves toward the region of greatest interest in the visual field.

See Sections 7.2 and 7.4.1 for details.

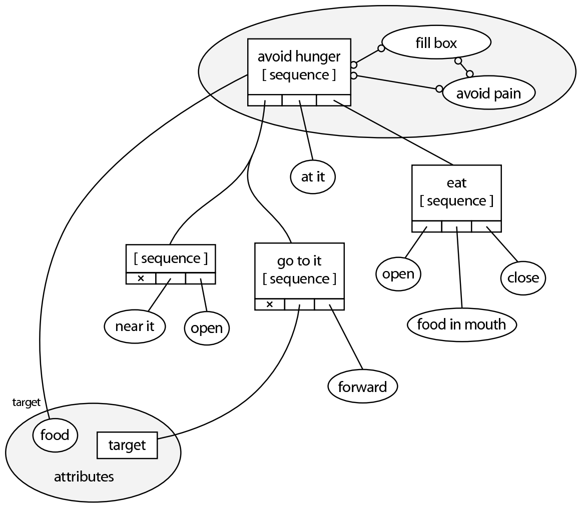


Figure A.3: Avoid Hunger Construction

Go to it relies on the vision system to find an object matching *target*, and moves forward when *target* is recognized (i.e. matches attribute agency states).

Avoid hunger loads food (*small* + *green* + *round*) into *target*, opens the mouth when near food, and eats when at food.

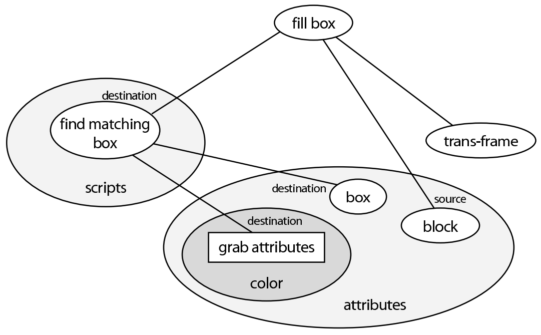


Figure A.4: Fill Box Construction

Fill Box sets *box* as the source object, and overrides the default trans-frame destination script with memory control to find a box matching the color of the object grabbed. See Figure A.5 and Section 7.4.4 for details.

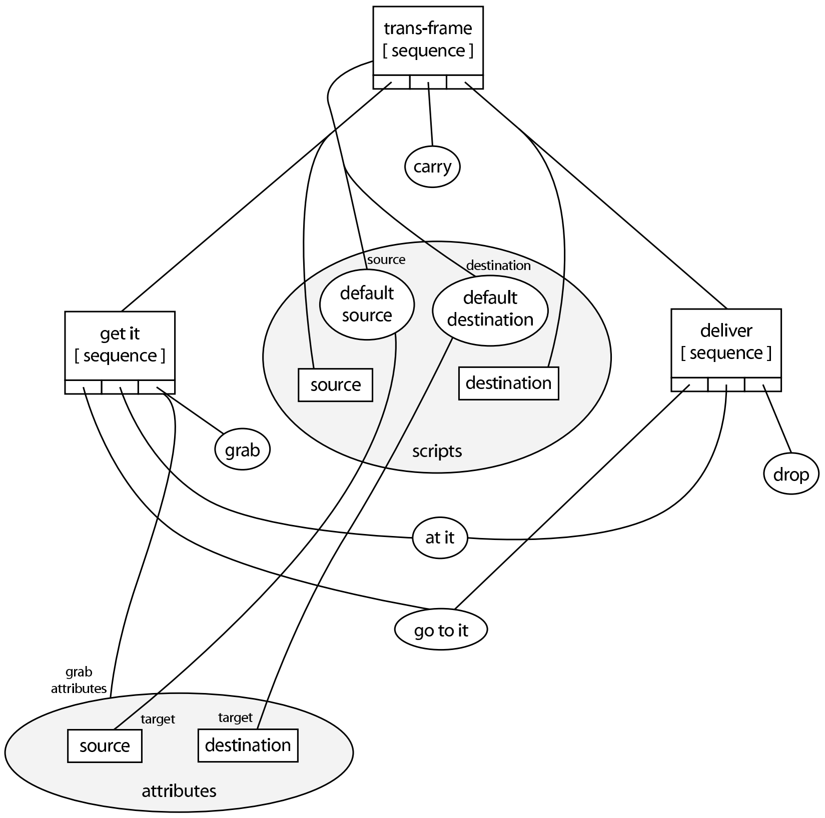


Figure A.5: Trans-Frame Construction

When active, the trans-frame weakly activates the *default source* and *destination* scripts into the *source* and *destination* script slots. The default scripts load the *source* and *destination* attribute short-term memories into the *target* attribute short-term memory. Other values in the script slots can override the defaults. See section 7.4.4 for details.

Bibliography

[1] William Bechtel and Adele Abrahamsen. *Connectionism and the Mind.* Blackwell Publishers, 1991.

[2] Valentino Braitenberg. *Vehicles.* The MIT Press, 1984.

[3] Rodney A. Brooks. *Cambrian Intelligence.* The MIT Press, 1999.

[4] A.G. Cairns-Smith. *Evolving the Mind.* Cambridge University Press, 1996.

[5] S Corkin. Lasting consequences of bilateral medial temporal lobectomy: Clinical course and experimental findings in h. m. *Seminars in Neurology*, 4:249–259, 1984.

[6] Joaquín M. Fuster. *Memory in the Cerebral Cortex.* The MIT Press, 1995.

[7] R. V. Guha and Douglas B. Lenat. Enabling agents to work together. *Communications of the ACM*, 37(7):127–142, 1994.

[8] J. Y. et al. Lettvin. What the frog's eye tells the frog's brain. *Proc. Inst. Radio Engr.*, 47:1940–1951, 1959.

[9] W. S. McCullough and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Math. Bio.*, 5:115–133, 1943.

[10] Marvin Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, 1967.

[11] Marvin Minsky. A framework for representing knowledge. In P. H. Winston, editor, *Psychology of Computer Vision.* McGraw Hill, 1975.

[12] Marvin Minsky. K-lines: a theory of memory. *Cognitive Science*, 4(2), 1980.

[13] Marvin Minsky. *The Society of Mind*. Simon and Schuster, 1986.

[14] Marvin Minsky. Logical vs. analogical or symbolic vs. connectionist or neat vs. scruffy. In P. H. Winston, editor, *Artificial Intelligence at MIT: Expanding Frontiers*, volume 1. The MIT Press, 1990.

[15] Marvin Minsky and Seymour Papert. *Perceptrons, Expanded Edition*. The MIT Press, 1988.

[16] C. R. Mooers. Zatocoding and developments in information retrieval. *ASLIB Proc.*, 8(1):3–22, 1956.

[17] Larry R. Squire and Eric R. Kandel. *Memory*. Scientific American Library, 1999.

[18] Mark J. Stefik and Stephen Smoliar. Four reviews of the society of mind and a response [fix eds]. *Artificial Intelligence*, 48:319–396, 1991.

[19] Shimon Ullman. *High-Level Vision*. The MIT Press, 2000.

[20] Matthew A. Wilson. Reactivation of hippocampal ensemble memories during sleep. *Science*, 265:676–679, 1994.

[21] Patrick Henry Winston. *Artificial Intelligence*. Addison-Wesley, 3rd edition, 1992.