NEURAL SCHEMAS: TOWARD A COMPREHENSIVE MECHANISM OF MIND

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Memphis

Thomas Lee McCauley

May, 2002

Abstract

McCauley, Thomas Lee. Ph.D. The University of Memphis. May, 2002. Neural Schemas: Toward a Comprehensive Mechanism of Mind. Major Professor: Stan Franklin, Ph.D.

Humans live in a sea of competing and complementary motivations. One reason that our behaviour is so complex and interesting stems from the hundreds of separate desires and goals that influence our actions. But we do not come into this world with these complex and often subtle motivators. Instead, humans learn to achieve or maintain intermediate goals that tend to facilitate the achievement of our base-level drives. We amass wealth, for example, because it facilitates the satiation of other drives such as hunger, security, or even procreation. Even though it can often be the case that a person will focus on the intermediate goal rather than the true priorities, these sub-goals are usually beneficial. Neural schema mechanism is a new autonomous agent control structure that makes use of both neural network and symbolic constructs to learn sensory motor correlations and abstract concepts through its own experience. The mechanism can also learn which intermediate states or goals should be achieved or avoided based on its primitive drives. In addition, a psychological theory of consciousness is modeled that allows the system to come up with creative action sequences to achieve goals even under situations of incomplete knowledge. The result is an architecture for robust action selection that learns not only how to achieve primitive drives, but also learns appropriate sub-goals that are in service of those drives. It does this in a way that is cognitively plausible and provides clear benefits to the performance of the system.

Table of Contents

List of Figures

"Practicality rules our perceptions. To survive, our tiny brains need to tame the blizzard of information that threatens to overwhelm us. Our perceptions are wondrously flexible, transforming our worldview automatically and continuously until we find safe harbor in a comfortable delusion." – Scott Adams, *God's Debris*

1 Introduction

As soon as a child is born it is given the daunting task of creating a reality. It must form the knowledge and goal structures that will dictate how it interacts with the world around it – and why. From relatively few initial clues each child must learn everything that it needs to survive and prosper. It needs to learn things like how its muscles work, how to communicate with its parents and others, and how to conform to the norms of its society. Some of this learning is just a matter of noting the common sensory results to actions performed under a given situation, simple muscle control, for example. Almost all learning beyond these first simple steps, however, requires that the child be able to judge not only what an action does, but also whether that result is a desirable one or not. Luckily for humanity, evolution has provided us with a set of innate sensory inputs that are pre-wired in our brains to give us pleasure, pain, happiness, sadness or any number of other feelings. Even so, there is more to what a child must learn than just which actions directly result in activation of one or more of the innate feelings. The child must also learn which environmental states that, although they do not trigger an innate response, nonetheless represent an increased likelihood of encountering a state that does trigger an innate positive or negative response at some point in the future.

To take the first steps in accomplishing this in an artificial system, a sufficiently general mechanism called schema mechanism created by Gary Drescher (1985, 1987,

1991) was extended to more closely approximate some human cognitive phenomena. The new extension is called neural schema mechanism due to its resemblance to connectionist architectures.

Even though the original schema mechanism, as described by Drescher, has been around for over a decade, very little experimentation with it or extension of it has occurred although there have been several uses of the knowledge representation proposed (Bickhard, 1997; Morrison, Oats, & King, 2001; Sun, Peterson, & Merrill 1999, 2001). This is probably due to its complexity and the processing resources required to implement the mechanism as it was originally formulated. The purpose of this dissertation is to describe a new implementation of the schema mechanism that extends its capabilities in important ways and takes steps to lessen the processing resources required.

The extensions described here are centered on an attempt to make the mechanism act more like a connectionist network (reasons for this desire are also presented). For example, the act of selecting a schema for activation is now done solely on the basis of activation levels and applicability as opposed to a set of rules that gauge the relationships between reliability, result-action correlation, cost of activation, and applicability. This modification changes the way that the mechanism acts in a subtle but important way related to how activation is spread through the network. Another important change that has been made to the system involves the redefining of how goal pursuit is accomplished. In the new version, the idea of maintaining a cost for the activation of each schema is replaced by the notion of spreading desirability initiated from a node in the network representing a goal state. We have also expanded the use of the broadcasting of goals to

more closely approximate a psychological theory of consciousness. In addition to providing a stronger psychological foundation, this allows the mechanism to discover novel solutions to otherwise insurmountable situations. All told, the neural schema mechanism takes steps toward the implementation of a comprehensive mechanism of mind that includes conceptual and sensory-motor learning, "consciousness," and a blending of logic with desires via a dynamic motivational system.

The proposed research aims to achieve several specific goals. In particular it will produce the following contributions to science:

1. The design and implementation of a new general-cognition architecture based on schema mechanism (Chapter 2: Mechanism overview and general methodology, Chapter 8: Neural Schemas).
2. Demonstrate the use and function of an implementation of Global Workspace theory as described by Baars (1997) (Chapter 8: Neural Schemas, Chapter 9: Results).
3. Show how the field of Artificial Intelligence can benefit from the implementation of psychological models and phenomenon (Chapter 3: How AI and Psychology benefit).
4. Empirical testing to determine how the new mechanism performs in a second environment with different types of learning tasks (Chapter 9: Results).
5. Suggest extensions to the resulting architecture that might further advance the fields of Artificial Intelligence and Cognitive Science (Chapter 10: Future Directions).

2 Mechanism overview and general methodology

One could envision neural schema mechanism as a network of interconnected nodes looking very much like a neural network. Some of the nodes in this network represent states of the environment, called item nodes; others, action nodes, represent actions that can be taken by the agent. Still others, called schema nodes, represent knowledge about the relationships between the environmental states and the actions. Finally, goal nodes, are present that act as the motivators of the system, trying to influence how activation is spread through the network in order to bring about or avoid bringing about the environmental state that they represent. All of these nodes are connected through various kinds of links. The many different types of links each have slightly different meanings and functions; some, such as action or goal-type links, simply serve as information pathways. Others serve as learning tools, collecting statistical information about the relationships between the nodes to which they are connected. In fact, it is a bit misleading to say that the schema nodes, alone, represent the knowledge of the system; it would be better stated to say that the schema nodes along with their result and context-type links actually describe the knowledge that is centered on the schema node. The upshot of all this is a mechanism that uses its own experience to learn about its world by creating new nodes and links that embody new knowledge about its environment and its own abilities to effect that environment.

Of course, no agent would be worth much if it were not able to pursue its own goals in an intelligent manner. Neural schema mechanism accomplishes this through the use of several different methods. Using experimental actions the mechanism learns over

time how to achieve the goals that it desires and to avoid the ones that it dislikes. In addition, the mechanism includes a "consciousness" module, consisting of attention and a global broadcast, which allows it to more efficiently utilize resources and discover novel and original solutions to daunting problems. Neural schema mechanism is not limited to the use of simple sensory and motor modules like those described in the later examples; such modules could well be very complex. Any of the advanced techniques from other more specific AI research could be used as senses or actuators.

The issue of importance for the moment, however, is how the various pieces of the neural schema mechanism come together to create an autonomous agent (Franklin & Greasser, 1997) that senses and acts within its environment in a meaningful and intelligent manner. Here is a first look at some of the elements that make up a neural schema agent.

2.1 Item Nodes

Every agent must have some way of determining the current state of its environment. In neural schema mechanism, environmental states are expressed through the activation of *item nodes*. Primitive item nodes, those that are built in at the time of creation of the agent, are attached directly to some sensory apparatus. Their activation can be set by the sensors to any value between –1 and 1, and is converted into a discrete state via a threshold function. If the node's activation is above the threshold, then the item node is considered to be *on*. If the node's activation is below some negative threshold then the item node is considered to be *off*. Otherwise the state is said to be unknown.

Item nodes can also be learned. These learned item nodes are created when the mechanism discovers a state of the world that is not expressed by any of the existing

items. The method of this discovery is based on the notion that the environment does not often change of its own accord; in other words, that things in the world tend to stay put for some period of time. For example, a person standing in one room of their house might look behind them and see a blue vase. Under most circumstances, looking backwards does not result in seeing a blue vase, but if the person has just turned around and seen the vase, then a repeat of the action is likely to result in the same relatively unusual vision. Neural schema mechanism uses this fact to create a synthetic item node that will, over time, approximate the conditions under which the unusual occurrence can be relied upon. Using this method, the mechanism can create concrete representations of abstract concepts as described in Section 8.2.2.

2.2 Action Nodes

Once the agent has perceived its environment it must have some way of acting upon that environment. This function is performed by *action nodes*. As with item nodes, there are both innate and learned action nodes. Primitive action nodes connect a neural schema agent to its actuators, while learned action nodes generally represent higher-level actions. One such higher-level action might be “ride-a-bike” as opposed to the numerous individual muscle movements necessary to actually ride the bike.

Even though the action nodes carry out the agent’s actions, an action node by itself cannot directly be executed. Instead it must be activated through the execution of a schema node for which it is the designated action. Each schema node can have one and only one action node associated with it.

2.3 Schema Nodes

Explicit knowledge of the agent's environment is expressed through *schema nodes*. Each schema node embodies the knowledge that a given action will have a specific result if it is executed under certain conditions. This is very similar to how a production rule or a "behavior" in a Maes Behavior Network (1989) might look; the primary differences being that a production rule does not usually designate an expected result and, for Behavior Networks, the action is simply the assumption of the result state. The result and context of a schema node are each made up of a set of item nodes to which it is connected via result and context links respectively, but the result is not assured as it is in Behavior Networks.

The links contain information relating to the relevance of the linked-to item with regard to the schema node's action. For instance, a result link states that the item node linked to is more likely to turn *on* (or *off*) when the schema node's action is executed than when the action is not executed. Note that this does not imply that the result is likely to occur, only that it is more likely to occur when the action is taken than when it is not. In a similar fashion, context links state that the results specified by the schema node are more likely to follow the action when the linked-to item node is *on* (or *off*).
Learning new schema nodes is the primary form of knowledge acquisition in the mechanism. The initial state of a neural schema network contains only primitive item, goal, and action nodes along with what are called "bare" schema nodes. A bare schema node is one in which there is only a link to an action node. Since action nodes cannot directly be executed, this is necessary to allow the system to perform any actions. It also gives the system the foundations upon which future learning can take place. When a

schema node is chosen for activation (randomly at first) the change of activations in the item nodes will tend to cause them, along with the just previously activated schema node, to be attended to by the attention mechanism. This, in turn, causes generic links to be created between the schema node and item nodes. These generic links will, from that point forward, maintain statistical information needed to determine the node's relevance to that schema node's action. For instance, if a potential result link (one that has a schema node as the input node and an item node for output) notes that its item node turns *on* more often than it turns *off*, then it will have a high positive relevance. Note that this statistic is not affected when the state of the item node does not transition. When the relevance of a potential result link is significantly positive or negative then a new schema node is created that is a duplicate of the existing one except that it has a result link noting the relevance of the particular item node to which it connects. Once a schema node has one or more result links, potential context links (those that have an item node as input and a schema node for output) note if its item node's state is a determining factor to the success of the schema node. For this, the link must maintain statistics on the success of the schema node when the item node's state was *on* as opposed to *off*. When a relevant item node is noticed through a potential context link, a new schema node is created with that item node connected to it via an appropriate context link. In this way the agent's knowledge of its environment is constructed through its own experience.

Schema nodes also keep track of their own reliability. In other words, given that the schema node's context is satisfied and the action is executed, what is the probability that the entire result set will obtain? The reliability of a schema node is a factor in

determining how much desirability flows through it and the likelihood that it will be chosen for activation.

The creation of new schema nodes eventually leads to the chaining of nodes where one schema node's results correspond to the context set of another schema node which, itself, feeds into a third schema node and so on until a goal state is reached. But with multiple paths to a goal or even to multiple goals, the system must have some way to judge which is the best path.

2.4 Goal Nodes

The motivations of a neural schema agent take the form of goal nodes. A primitive goal node is analogous to basic drives; its purpose being to influence the mechanism in such a way as to bring about its goal state, however that might be accomplished. Whether the goal node is primitive or learned, the method for bringing about its goal state is the same, namely the spreading of desirability through the network.

Desirability is spread like activation except that it flows backwards through the network. Desirability attempts to measure the usefulness or importance of activating a given schema node with respect to achieving the agent's current goals. There are two primary ways that desirability changes the behavior of the agent. First, the desirability on a link modifies the weight on that link so as to increase or decrease the amount of activation that gets delivered to the output node. Secondly, the desirability on a particular schema node is used as a determining factor in selecting which schema node to activate on any given cycle.

A primitive goal node in the new mechanism will have an intrinsic primitive desirability value provided at design time. Even though these nodes propagate

desirability constantly, the amount sent out is determined by the degree to which the goal state holds in the external or internal environment. A goal node for eating, for example, would need to vary its desirability output based on the agent's hunger.

New non-primitive goal nodes are created whenever a new schema is created whose result set is novel. These learned goal nodes, therefore, represent the desire to attain that state. The question, of course, is how to determine whether this arbitrarily abstract state should be generally sought or avoided. The mechanism accomplishes this through the concept of delegated desirability. Learned goal nodes keep track of the difference between the highest desirability value of any applicable schema nodes when the goal node's state is *on* and when it is *off*. The delegated desirability of the goal node is, therefore, a function of that difference and will, over time, acquire appropriate values for the goal state.

Obviously there is more to this model than just the creation of a mechanism to intelligently perform actions. While that element is not insignificant, there is also the desire to explore the scientific realm of how human minds work. The two sides of science and engineering need to feed one another in a way that has the potential of reaching an outcome that could not have been attained from either side alone.

3 How AI and Psychology benefit

Perhaps the greatest single driving force for this research is simple curiosity, a desire to discover how minds work. This desire coupled with a belief that true understanding accompanies implementation suggests that an artificial version of promising theories and psychological phenomenon be created to whatever degree possible. This attitude fosters a close interaction between scientific theory and engineering.

Arguably, the ultimate goal of Artificial Intelligence is to create human level or better cognition in a man-made system. One strategy to achieve this goal is to focuses on using what we know of human minds (and brains) to improve the performance of what we can build which, in turn, improves our theories and knowledge. The science part of this loop would look something like the following, taken from (Franklin, 1997b):

1) Design a cognitive agent architecture.
2) Implement this cognitive architecture on a computer.
3) Experiment with this implemented model to learn about the functioning of the design.
4) Use this knowledge to formulate the cognitive theory corresponding to the cognitive architecture.
5) From this theory derive testable predictions.
6) Design and carry out experiments to test the theory using human or animal subjects.
7) Use the knowledge gained from the experiments to modify the architecture so as to improve the predictions of the theory.

The engineering side would be something to this effect:

1) Design a cognitive agent architecture.

2) Implement this cognitive architecture on a computer.

3) Experiment with this implemented model to learn about the functioning of the design.

4) Use this knowledge to design a version of the architecture capable of real world (including artificial life and software environments) problem solving.

5) Implement this version in hardware or software.

6) Experiment with the resulting agent confronting real world problems.

7) Use the knowledge gained from the experiments to modify the architecture so as to improve the performance of the resulting agent.

By linking these two loops together and allowing each one to feed into the other, it is hoped that better cognitive theories and better artificial agents will emerge. The neural schema mechanism is an example of one iteration of this strategy. The original schema mechanism, itself built to model a psychological theory (Piaget, 1952, 1954), has been revised to include additional functions observed and theorized about in humans.

The interaction between computer science and cognitive science employed in this research has yielded a mechanism that can further both areas in significant ways. With continued experimentation this mechanism should produce predictions that can be empirically tested in humans. If the predictions bear true and are novel, then a cognitive theory can be formulated that encompasses the mechanisms nuances. Non-novel predictions serve to bolster corresponding theories. Even if the predictions are shown to be wrong, the cognitive science community will have benefited from this knowledge and direction will have been set for future implementations.

The AI community, for its part, gains a new set of algorithms and methods upon which further extensions can be based that enhance the mechanism regardless of the

cognitive aspects. Several areas of the neural schema mechanism, in particular, will hopefully provide other AI researchers with new techniques not seen elsewhere. For example, this system utilizes spreading activation coupled with spreading desirability along the same links to accomplish goal directed behavior. This method may prove useful in other connectionist architectures. Other aspects of the mechanism created with human psychology in mind should also be looked at as alternative approaches to common AI issues. Most notable among these aspects is the makeup and use of attention and "consciousness" to enhance the behavior and learning of an autonomous agent. Two other projects, CMattie and IDA, from the "Conscious Software Research Group" at the University of Memphis have also delved deeply into this area (Franklin, 1997a; Franklin, Keleman, & McCauley, 1998; Ramamurthy, Bogner, & Franklin,1998). Even if one ignores the correspondence to psychology, the algorithms developed here can be put to good use in other systems that require sensory filtering via an attention-like mechanism and the ability to solve non-routine problems for which a path to some goal state is not obvious or derivable by simple action-result chaining. To assist the computer science world in using these methods, the code for the neural schema mechanism will be available for educational use without charge.[1]

[1] Complete details are listed in Appendix D

4 Why neural schemas?

Perhaps the question would be better phrased as 'why should schemas be neural?' The primary answer to this is rooted in the second fundamental AI debate (Franklin, 1995). This debate weighs the merits of classic symbolic AI versus a connectionist model. On the symbolic AI side, the argument is that, while connectionist systems can perform the same functions as a symbolic system, they do not add any additional capabilities or knowledge. The connectionist side, on the other hand, states that its models function in a fundamentally different manner and that they have made significant advances to cognitive science. For an in-depth discussion of this debate, see Franklin (1995, p. 141-164).

The aspect of this argument that is particularly relevant here relates to whether a connectionist model can produce solutions in situations where a symbolic system would not function well or in a manner that increases the knowledge of the situation in a way that a symbolic method could not. With regards to differentiating the original schema mechanism with the new implementation, there is reason to believe that the answer to this question is "yes." The first evidence for this position lies in the inherently "soft" way that a connectionist system functions. "Soft" here is taken to mean that there are no hard rules of the form [if A then B], as you would find in a classic symbolic system. In fact, as David Chalmers (1990) points out, symbolic AI can only have hard rules and any ambiguity must be in the form of exception rules. The result for symbolic systems is an exponentially proportional decrease in processing performance with every increase in complexity. For this reason, a symbolic approach does not usually scale well to complex

problems. A connectionist model, on the other hand, can produce "good enough" solutions even in the midst of a complex and dynamic environment. The reason for this ability comes from the fact that a network with continuous activation performs a portion of the calculations necessary for acting at any given moment in advance. To illustrate this point let's look at an example from Horgan and Tienson (1989). Imagine if you will, that you are a point guard on a basketball team who is dribbling down the court and has to decide whether to take the shot or pass the ball off. An almost infinite number of factors come into play just to make this decision. Where are you on the court? Where are your teammates? Where are the opposing players? Is your balance and body position correct for a shot? How about for a pass? Who is guarding you? Who is guarding your teammates? Who has the hot hand? What is the score? How much time is left in the game/period? What are the offensive and defensive strengths of all the other players on both teams? Is there one of your teammates whose grandmother came to see this game and hasn't had a lot of playing time? Etc. All of this must be considered and acted upon in milliseconds. A symbolic system would need to perform the full calculation for all of these aspects between the moment that the decision is instigated and the action is necessary. Therefore, a symbolic system will be increasingly hard pressed to act with each additional feature.

A continuously active connectionist network, however, only needs to factor in the most recent changes in the environment in order to make a decision; all other factors are already represented as lingering activations in nodes or groups of nodes. In other words, the state of the environment and agent is maintained constantly from one moment to the next and effects every decision at every moment. The connectionist system doesn't need

to figure out how all the old states effect each other at each choice point; it only needs to add the effects of the latest changes in states to the system as a whole. This is not to say that the symbolic system could not come up with a solution; it may even be a better solution. The point is that the connectionist system can produce a solution in adequate time that, while not necessarily optimal, will suffice for most cases.

Keep in mind that the argument just presented relates specifically to the "old" classical forms of symbolic AI. Many newer models involving fuzzy production rules and probabilistic rule firing are much less susceptible to the brittleness of classic mechanisms. One might say that the mathematical equivalence of symbolic and connectionist systems (Garzon & Franklin, 1991) is beginning to converge. However, there is still one feature of connectionist systems that is difficult to realize in symbolic models: implicit knowledge. Specific information in a neural network, for example, is not likely to be represented by a single node. Instead it has a sort of "virtual" existence as weights on links connecting nodes. This gives a connectionist system the feeling, at least, of having a more robust knowledge representation that can shift and change within the high-dimensional space created by the network without the need to add additional nodes. With that said, it should also be noted that neural schema mechanism is more like a symbolic system in this regard and, while the knowledge representation is somewhat distributed, it is still primarily a system of explicit knowledge.

The use of connectionist properties in the schema mechanism does, however, allow for a reduction in the number of calculations that need to be performed within each time slice. This is accomplished, primarily, by reducing the branching within the network. Instead of having a fully connected network, as Drescher describes in the

original mechanism, neural schema nodes are connected only when they have been in the spotlight of attention together (explained in more detail later)[2]. The result of this is that only nodes whose activations co-occur within a time window are connected. While this may marginally slow the learning rate of the agent, it lowers the amount of work required of a system running such an agent at any given time step and decreases the overall size of the data structures required.

There is one caveat regarding the implementation of the neural schema mechanism as a connectionist network. A good portion of the speed benefits of a connectionist system comes from its parallel nature. While a symbolic system must, for the most part, be executed on a serial platform, connectionist systems can, theoretically, utilize a separate processor for each node in the network. Much of this benefit, however, is mitigated when a connectionist system is simulated on a serial computer such as a standard PC as is the case with this implementation. For this reason the full potential of this mechanism has not yet been realized in the current implementation. Even so, the arguments just listed for the use of a connectionist framework still hold even if they are less visible due to present limitations.

[2] The use of an attentional mechanism to reduce the computational load of the system was also done by Foner and Maes (1994), although their algorithm was significantly different from the one utilized here.

5 Overview of Drescher's schema mechanism

The stated purpose of the original schema mechanism was to reproduce parts of the cognitive development of children as described by Jean Piaget (1952, 1954). To this end, Drescher created a mechanism that could learn in a variety of different ways and form new concepts upon which additional learning could take place.

The schema mechanism consists of three different types of structures: schemas, items and actions. The schema is the main structure within the mechanism (hence the name). It is made up of a list of contexts, a list of results, and an action. In its list of results, a schema will keep up with those items whose state the schema expects to turn on or off as a result of the action. The context list, similarly, contains those items which must be on or off in order for the results to follow that schema's action. A schema is intended to state a conjecture about the agent's environment relating the results to the action given the contexts. From an individual schema's perspective, it states that if all of its context slots (filled with items that should be either *on* or *off*) obtain, and its action is performed then its result slots (also filled with items that should be either *on* or *off*) will obtain. Anytime that a schema is applicable (all of its contexts obtain) and its action is performed, then it is considered to have been activated. Note that a schema does not have to be explicitly chosen for activation to be active. This allows for implicit activation, which facilitates the ability for multiple schemas with the same action to all learn about the results of that action simultaneously.

Each schema maintains a number of statistics that allow it to learn new schemas and to revise its own accuracy. First, each schema keeps track of its correlation. The

correlation is the ratio of the probability that the schema's result obtains when activated to the probability that the result obtains when not activated. Essentially, this measures the importance of the schema's action to obtaining its result. Each schema also calculates its own reliability. This is the probability that the schema's result obtains when the schema is activated. In other words, how often the schema's activation causes all of the schema's results to occur. A more specific reliability measure, the local reliability, is calculated and used to determine when a synthetic (learned) item (described below) should be created. Local reliability is the likelihood that, if the schema has recently been activated successfully, then a repeated activation will also succeed. The average duration time for the schema's action is also maintained along with the average negative cost of activating the schema.

In addition to the standard lists of contexts and results that each schema contains, there are also extended context and extended result lists made up of information relative to every other item in the system. Each slot in these lists holds statistics for the correlation between the schema and the item. These extended lists are primarily used for determining when a new schema should be created. Based on the extended context and result lists, the mechanism can create new schemas that make new assertions about the state of the environment. The first step in this process begins with a 'bare' schema. A bare schema is one that contains only an action; its context and result lists are empty. In the extended result list, the schema keeps track of the positive and negative-transition correlation for every item in the system. A transition correlation is the likelihood that a given item will change its state to *on* (positive) or *off* (negative) following the schema's activation. If either the positive or the negative-transition correlation for an item

becomes sufficiently greater than the other, then a new schema is created with that item added to the result list with its inclusion state matching the greater transition correlation. If the set of results denoted by the new schema's result list is novel (no other schema has that result), then a new action is created with this result as its goal. Also, a bare schema is created that contains this new action. The reason for these additional structures will be discussed below in the description of composite actions.

Once a schema has been created that asserts a result for an action, the schema's extended context list can begin to discover the circumstances under which the assertion holds. Each extended context item must maintain its relevance to the schema's result when the schema is activated. Relevance is defined as "the ratio of the probability that the schema will succeed if the schema is activated when the slot's item is on, to the probability of success if that item is off when the schema is activated" (Drescher, 1991). Once again, if either of the two probabilities is sufficiently greater than the other, then a new schema is created with this item added to its context list. Again, which probability is greater determines the item's inclusion state, *on* or *off*, in the list.

The next major part of the mechanism is the item. An item represents a state of the environment. That state can either be *on*, *off*, or *unknown*. In the initial setup of an agent, the items that it is aware of, called primitive items, consist of direct sensory elements. For example, a single primitive item whose state is determined by a sensory apparatus of some kind could represent that the agent sees light. Learned items are also created which represent a state in the environment that persists even without direct evidence of such via the senses. When a schema is not generally reliable but its local reliability is high, then it is a candidate for synthetic item creation. Learned items are an

important part of the discovery process within the mechanism. They establish the counter-factual assertion that a given action would produce a given result if that action were taken under a specific set of circumstances. It is not necessary that the action actually be performed for the item to be considered to be *on*. Other schemas can then use this learned item in their results or contexts. An example might help to show the importance of this element. Suppose that an agent is in a dark room. The agent has noticed that, sometimes, when it takes a step forward, that it runs into something. In general, the act of taking a step forward does not usually result in the agent running into an object. However, if it has just tried to step forward and stubbed its toe, and then repeats the same action, it will stub its toe again. A learned item is created with this schema as its host. The schema is said to reify the item. In other words, the schema designates the conditions under which the item's state is *on* (or *off*) even though the agent doesn't initially know what those conditions are. Further context spin-offs of the schema will refine its ability to note the state of this learned item. The mechanism can treat that item as an object such that it does not need to actually move forward while at the special location, but can create chains of schemas that move the agent around the object without running into it. Even though this is a simple example, it demonstrates how learned item formation can change the behaviour of the system in a very important way. From this point, schemas can be created whose contexts or results contain the newly created item node.

The iterative use of this item creation method results in the theoretically infinite abstraction of concepts. Along with the continuous refinement of existing schemas and the creation of new schemas, the mechanism is constantly modifying its approximation of

its environment. It is important to note that while the mechanism can create an item and treat it as an object, it can never *define* the object; it can only approximate its verifying conditions to a greater and greater degree of accuracy.

The final element in the schema mechanism is the action. As with items, there are two kinds of actions in the system, primitive and composite. Primitive actions are those that are innate and are usually connected directly to some actuator within the agent. A composite action is one that is learned. The conditions for learning such actions was briefly described above, but to recap, a composite action is created when a new schema is produced that has a novel result. The action contains a list of items that represent its goal state, much the same as a context or result list. Also, the action initializes an action controller that maintains a list of all of the schemas in the system and their proximity to the goal state. When a composite action is performed, its controller selects the schema that is closest to the goal state and activates it. In this way, a composite action can be instigated in a manner tailored to the environmental situation and need not be performed in exactly the same way every time. Since an action may require multiple schemas to be activated in sequence, and the schema, which performs that action, must continually compete for activation, it is possible that an action will not complete before a different schema and, therefore, a different action is selected. This provides for a form of multiple goal pursuit where schemas are alternately being activated and converge on both goals. Each action keeps track of the average time that it takes to complete and uses this value to determine when the action has failed.

An additional aspect of goal pursuit is the way that schemas are selected for activation when the mechanism is attempting to attain a specific goal (as opposed to

activating a schema simply for its learning potential). Schemas are selected for activation based on their applicability and the importance of their activation. Given that a schema is applicable, it asserts the importance of its activation based on a value that has been propagated to it through chains of schemas. The propagation begins in items that have been designated as goals and proceeds through chains of schemas. Each item designated as a goal is given a value representing the importance of that item turning *on* (or *off* if a negative value). These values are then sent to all schemas that have those items as results. The receiving schemas make note of the value and pass that value, along with an accumulated cost, on to all schemas that have its complete context set as part or all of their result set through the items that the two schemas have in common. Some simple rules govern which values to pass along in the case of the intersection of two broadcast paths. When the propagation of values is through, each schema on a path to a goal item will have an importance value that represents the highest valued goal item to which it chains. The importance value then becomes the deciding factor in selecting which of the applicable schemas to activate in a given cycle.

The values given to items, themselves can come in three varieties: primitive, instrumental, and delegated. Primitive value is the value a non-learned item is given at the onset of a run. This is like primitive desires in humans such as hunger and sex; they are provided at birth thanks to evolution. Instrumental value is that value passed to an item via the propagation method described in the previous paragraph. It can change from one cycle to the next and says nothing about the item's long term or general value. Instead, instrumental value represents the item's importance in reaching a desired goal at that specific moment. Delegated value, on the other hand, accumulates over time and is

based on the likelihood that the item is a key element along some path to a positive valued goal. It is through the use of delegated value that the mechanism learns which items should be generally sought or avoided regardless of whether they achieve some currently pursued goal.

Drescher's schema mechanism is a robust architecture for learning and action selection in autonomous agents. While it is not intended as a complete model of mind, it comes close in many areas. In addition, Drescher, apparently unknowingly, includes elements in the mechanism that resemble the beginnings of an implementation of a particular theory of consciousness. The new schema mechanism, described below, attempts to fill out the architecture in ways that increases its correspondence with psychological phenomena such as habituation and priming, and includes a theory of consciousness (Baars, 1988, 1997).

6 Comparing Drescher's mechanism and neural schemas

In addition to modifying Drescher's original mechanism, a different environment was used as the primary testing platform for neural schemas. A representation of Drescher's micro-world is shown in Figure 6.1. His environment consists of a 7 x 7 grid within which the agent (or child) is located. The agent's location is constant and is represented by a bird's-eye-view of a child's head (middle bottom). The agent has a visual sense represented by the lighter colored areas of the grid and can be moved in any of the four ordinal directions within a small range. The dark gray areas of the grid are outside of the agent's visual field. The light gray areas are within the agent's visual field but are not in clear focus. Objects in this area are represented as blobs as can be seen in the top right

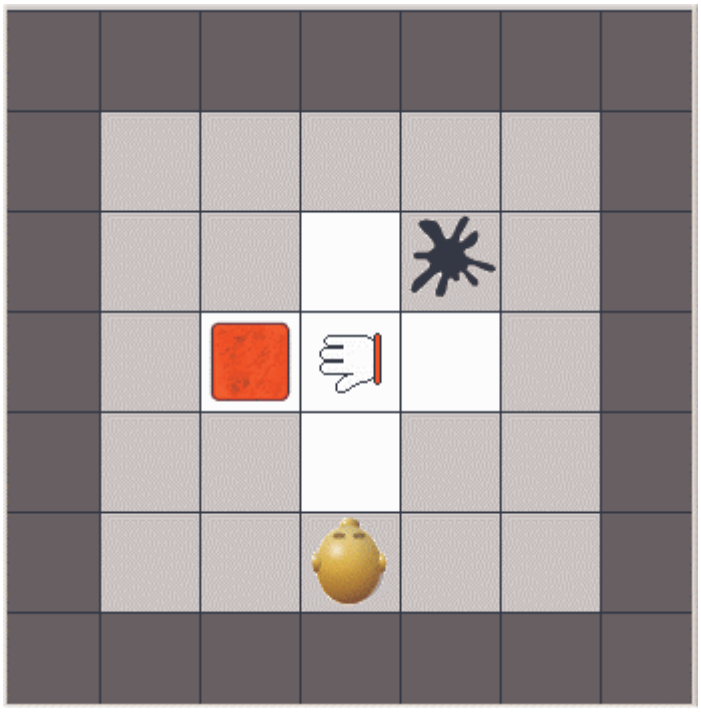

Figure 6.1: Representation of Drescher's original environment

quadrant of Figure 6.1. From the agent's perspective, the only thing that can be gleaned from these blobs is that there is something in its visual field at that location. The white area in the center of the visual field is called the foveal area. Items in this area can be fully viewed. Each of the five locations within the foveal area has a set of sixteen detailed visual elements that it can recognize about items in that square. There are also two objects in the environment with which the agent can interact. These objects can be moved if the agent grasps them with its hand or if it pushes one object with a grasped object. The agent's hand can also move in any of the four ordinal directions within a range and can grasp an object (or close its hand if no object is available) and un-grasp (or open its hand). For a complete description of the original environment see (Drescher, 1991).

The neural schema mechanism produced similar results to the original mechanism within this micro-world. Early in a run neural schema mechanism learned the same concepts is approximately the same order as the original. Almost without exception the first concepts learned are the fact that the "grasp" action results in the hand being in a closed state and that the "un-grasp" action results in the hand not being in a closed state. The system also quickly begins to recognize that certain movement actions of the hand and eye result in the position of the hand or gaze transitioning into new states. This is denoted with new schema nodes that represent that performing the "move hand forward" action, for example, results in the position state of 2, 3 turning *on*. Eventually the mechanism would reach a state where learning was occurring only very rarely and almost all of the agent's possible actions and states had been completely. In other words, to use our previous example, there would be a schema node that represented the fact that

performing the “move hand forward” action while hand position 2, 2 (middle position) was *on* resulted in the position state of 2, 3 turning *on*. A complete mapping consists of all possible movement and corresponding results of the type just described. The similar performance within this small domain is not surprising due the fact that almost all of the enhancements to the mechanism are designed to increase its performance in environments that require differentiation of more complex goal-oriented situations.

Some encouraging results have been noted with respect to the use of the attention mechanism and consciousness model. The attention mechanism has consistently reduced the linkage between nodes while maintaining relevant connections. For instance, early in a run the schema nodes for the “grasp” and “un-grasp” actions are often connected only with the item nodes for the “hand closed” and “hand grasping” states. Learning occurs as it should without burdening the system with unnecessary connections.

There was, however, a substantial difficulty with using Drescher’s micro-world to

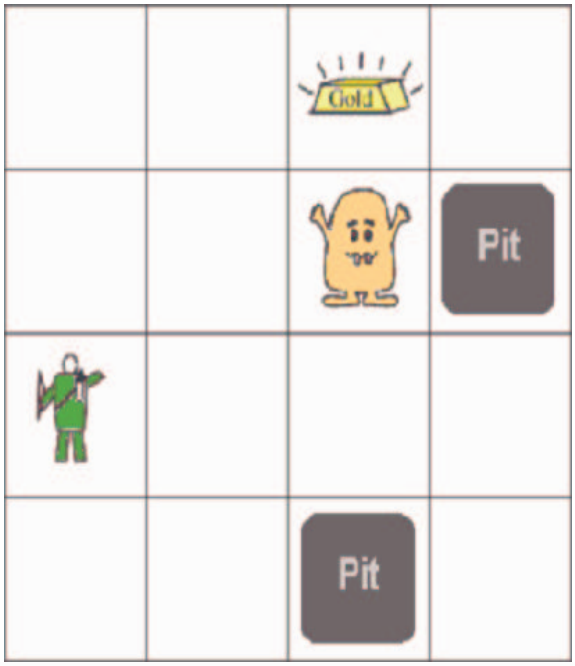

Figure 6.2: Wumpus World (Russell & Norvig, 1995)

test neural schema mechanism: the lack of clearly defined goals. To remedy this, a different environment (Figure 6.2), called Wumpus World (borrowed from Russell & Norvig, 1995), was chosen as the primary test bed. In the Wumpus World, the agent, in this case a "hunter", can sense the glitter of gold, the breeze from a pit, and the stench of the wumpus if he is within one square of the object. The hunter can also sense the gold, being eaten by the wumpus, and falling in a pit if he is in the same square as the object. The agent also senses his position, whether his is carrying the gold, if he bumps into a wall (directionally significant), and if he has exited the cave (grid). The hunter has the ability to move in any of the four directions (north, south, east or west), grab for the gold, and climb out of the cave. The objective for the agent is to locate the gold and find his way out without being eaten by the wumpus or falling into a pit.

There are several trade-offs that were made as a result of choosing this new environment. The most obvious negative trade-off is the fact that the number of initial sensory items is less in the Wumpus World. Therefore, it should be theoretically easier for the agent to learn a complete mapping of its environment via schema structures. The main purpose of the new mechanism, however, was not just how well it could learn about its surroundings, but also how well the system could ignore irrelevant items. To this end, the Wumpus World environment contains a sufficient number of generally applicable sensory item nodes that may or may not be relevant to any given situation. The number of sensory items that tend to be relevant to a particular action at any specific moment is approximately equal for both Drescher's micro-world and Wumpus World. For this reason, the Wumpus World is comparable to Drescher's micro-world for the purposes of testing relevancy learning.

In addition, Wumpus World is superior to Drescher's micro-world when it comes to the testing of goal-oriented actions. Not only does Wumpus World provide for specific goals for the agent to accomplish, but it also contains multiple goals that will alternately conflict and support one another depending on the situation. Since this was the principal focus of neural schema mechanism, it was decided that Wumpus World would be preferable to Drescher's micro-world for testing purposes.

7 Related Work

In addition to Drescher's original schema mechanism, several other technologies have lent parts of themselves to this research in the form of ideas or methods for accomplishing certain tasks. This section will describe the systems that contributed to the neural schema mechanism.

7.1 The Pandemonium Association Engine

This architecture is based on a psychological theory called Pandemonium Theory (Selfridge, 1959) that was used to describe human perception. Later, John Jackson (1987) presented it to the computer science community in an extended and more concrete form (Franklin, 1995) that makes it useful for control of autonomous agents.

In Jackson's version of Pandemonium Theory, the analogy of a sports arena is used. The arena consists of stands, a playing field, and a sub-arena. It is also populated by a multitude of "demons," each a simple agent. Some of the demons will be on the playing field doing whatever it is they are designed to do; these demons are considered "active." The rest of the demons are in the stands watching the playing field and waiting for something to happen that excites them. Of course, what is exciting may be different for each demon. The more exciting the action on the field is to any particular demon, the louder that demon yells. If a demon yells loudly enough, it gets to go down to the playing field and become active. At this point, it can perform its function. Its act may excite other demons, which may become active and excite yet other demons, etc.

Which demons excite which other demons is not a random matter; each demon has associations with other demons that act much like weighted links in a neural network.

The activation level of a demon (a measure of how loudly it is yelling) spreads down the demon's association links and, therefore, contributes to the activation level of the receiving demon. In addition, these associations are not static. Whenever a demon enters the playing field, the sub-arena creates associations (if they do not already exist) between the incoming demon and any demons already on the field. A strong output association and a weaker input association are created between the demons currently on the playing field and the arriving demon. This maintains a kind of sequencing. For example, demon A precedes demon B in a series. Demon A will have a strong association with demon B, allowing A to activate (excite) B, and B will have a weaker association with A so that B will not reactivate A but still maintains a link. The actual strength of the associations depends on a gain value that the sub-arena calculates. The gain is intended to be an estimate of how well the whole system is doing at any given time. In addition to creating these new associations, existing association strengths between demons on the playing field increase (or decrease) at each time step based on the gain value. Also, multiple demons that have strong associations with each other can be grouped together, to create a single new demon called a concept demon. From the moment of their creation onward, these concept demons act almost like any other demon in the system. They differ in that the decay rate of their associations is less, and the amount of time that they spend on the playing field at any one calling is increased.

In the stadium metaphor, the sub-arena is where all of the processes are carried out that are needed for the basic function of the architecture. In addition to the calculation of the gain, an example of a process that the sub-arena would be responsible for is the act of actually passing activation down a link. The sub-arena also performs the

actual input and output functions of the system as well as most of the automatic maintenance functions. It calculates the gain; a single variable intended to convey how well the agent is performing, although Jackson did not specify a mechanism for such an assessment. Surely the assessment must be both domain dependent and goal dependent. Since the gain determines how to strengthen or weaken associations between demons, how this judgment is arrived at, and how the goal hierarchy is laid out is of considerable importance. The agent accomplishes goal directed behavior only by an *accurate* assessment of its moment-to-moment status. For humans there is a complex system of sensory labeling and emotional responses (Demasio, 1994; LeDoux, 1989; Panksepp, 1995), tuned through evolution, which allows us to determine our performance based on currently active goal contexts.

The current goal context of this system changes dynamically. It can be thought of as emerging from the demons active at a given time. Some high-level concept demons can remain on the playing field for quite a long time and, therefore, influence the actions of the whole agent for that time. An example of such a high level demon might be one that tends to send activation to those demons involved in getting some lunch. Multiple goal contexts can be competing or cooperating to accomplish their tasks.

Even though the metaphor of an arena filled with demons is used to describe Jackson's model, it is very much a connectionist network that passes activation to nodes over weighted links and performs a summation at the node. What primarily differentiates this mechanism from standard neural networks is the method with which it learns. The system can begin with few links and learn which demons should be associated with one another and, over time, how much. The real issue, of course, is the ability of the system

to select appropriate actions for an autonomous agent[3] as Jackson intended. To date, Jackson has demonstrated that his system can learn to recognize sensory patterns in a simple environment and act based on these patterns based on a one-dimensional gain. What is unclear is whether the system would be able to function in a situation that had more than one conflicting goal. The single valued gain does not allow differentiation between degrees from one goal to the next. Even so, the Pandemonium Association Engine provides key features of the neural schema mechanism that will be discussed later. In particular, the notion of information being gleaned from the co-occurrence in time of active concepts is a theme that will be revisited several more times. Of course, this notion is not new with Jackson and is generally attributed to Hebb (1949). However, Jackson's particular use of Hebb's Rule inspired several areas of neural schemas.

7.2 Behavior Networks

Another mechanism that bears some responsibility for particular architectural features in the neural schema mechanism is Pattie Maes' behavior network (1990). This is also the mechanism that bears the closest resemblance to neural schemas mainly due to its use of preconditions and add/delete lists for each behavior[4]. Preconditions, here, are very similar to Drescher's context lists and Maes' add/delete lists describe the results of activating a given behavior.

Each behavior in Maes' mechanism consists of preconditions, a list of additions, a list of deletions, and activation. With the exception of the maintaining and passing of activations, this is very much a classic AI system that runs on production rules. A behavior would be something like `pick-up-rock` that would have `hand-empty` as a

[3] For a complete definition of "autonomous agent" see (Franklin & Graesser, 1997).
[4] Maes calls her behaviors "competence modules."

precondition, `rock-in-hand` on its add list and `rock-on-ground` on its delete list. The production rule equivalent would be something like, "if `hand-empty` then `rock-in-hand` = true, `rock-on-ground` = false." When a behavior's preconditions are true then that behavior is said to be executable. It is not the case, however, that each behavior has some action that it performs and, separately, has lists of expected results as would be found in schema mechanism. The action of activating a behavior is that the facts listed in its add list are put forth as true states of the environment and the facts listed in its delete list are removed from being true states of the environment.

Once we add other behaviors to the system, we need a way of selecting which behavior to run at any one time. This is done by, first, connecting the behaviors to each other via predecessor, successor, and conflictor links. These links are dictated by the behaviors themselves and turn out to be, in the original mechanism, nothing more than convenience pointers that keeps the system from having to search all the behaviors at each time step in order to determine where to pass activation. While this feature is not trivial, it does not change the output of the mechanism, only its speed. The following explanation of how these links are created is borrowed from (Franklin, 1995).

> If a competence module X will add a proposition b, which is on competence Y's precondition list, then put a successor link from X to Y. There may be several such propositions resulting in several links between the same nodes. Next, whenever you put a successor going one way, put a predecessor going the other. Finally, suppose you have a proposition m on competence Y's delete list that is also a precondition for competence X. In other words, X wants m and Y wants to

get rid of m. In such a case, draw a conflictor link from X to Y, which is to be inhibitory rather than excitatory. (pp. 247 - 248, See Figure 7.1)

Now that the network has been created there must be sources of activation that will power the interactions of the behaviors and the actions of the agent. Those sources come from the environment and goals. A set amount of activation is added to any behavior for which a precondition is being satisfied by an environmental state. Likewise, any behavior that can satisfy a goal state is given a certain amount of activation. Together, these two factors assure that the system acts in a way that is both goal oriented

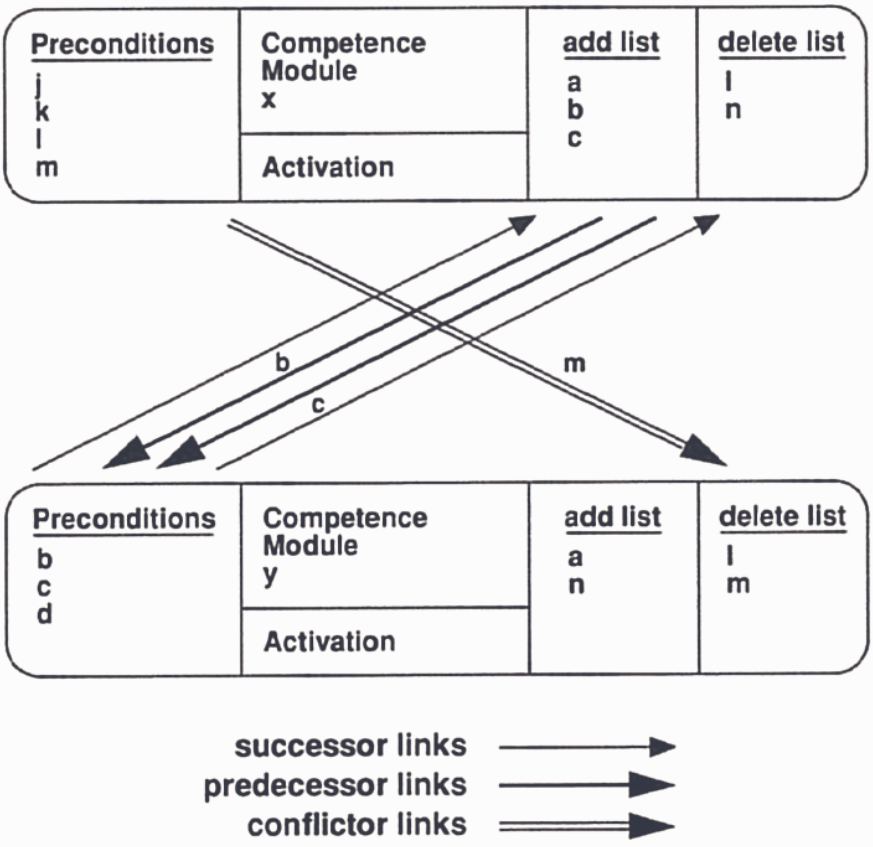

Figure 7.1: Links in a Behavior Net (Franklin, 1995)

and situation relevant. Finally, each behavior sends some of its activation through predecessor links to other behaviors whose preconditions it can help satisfy and sends some of its activation through successor links to behaviors whose execution would help to satisfy its own preconditions. Negative activation is sent down conflictor links to behaviors whose execution would keep one of its preconditions from being true.

After all of the activation has been added to the network and passed through the necessary links, the total activation of the network is normalized to keep the total activation of the system constant. Now a behavior is selected for execution. A behavior is selected if all of its preconditions are met, it is over an activation threshold, and it is the behavior with the greatest activation among the behaviors that meet the first two criteria. If no behavior is above threshold and applicable, then the activation threshold is lowered by 10% and the process begins again. Otherwise, the chosen behavior executes, its activation is reset to 0, and the activation threshold is returned to its default value. The point of describing this algorithm, aside from a general attempt to convey an understanding of the mechanism, is to allow for comparison with a similar algorithm that will be laid out while describing neural schemas.

The specific similarities and differences will be discussed in greater detail later. For now, however, it should be easy to recognize how schema mechanism and behavior networks share the common notion that a base unit in the mechanism should consist of a context-action-result triple. Incidentally, there is a fairly large body of knowledge that has been built up around a mainly psychological examination of this type of model called "discrimination learning." Discrimination learning can be described broadly as the study of how thinking entities, such as insects or animals, learn to discriminate between stimuli

(Langely, 1987; Rescorla & Wagner, 1972; Spence, 1936). For example, how does a sparrow learn to recognize a camouflaged butterfly against the background of tree bark? By necessity, experiments of these abilities require that each stimulus tested have some degree of positive or negative reinforcement. Neural schema mechanism could be seen to be a computational implementation of discrimination learning. The main difference, however, is in the fact that neural schema mechanism is chiefly concerned with learning how the agent's actions effect the environment instead of how the agent differentiates between environmental states.

7.3 Construction Integration Model

The final model that will be mentioned as it relates to neural schemas is Walter Kintsch's Construction-Integration (CI) model (1988, 1998). The similarity between these two works is as much in spirit as it is in substance. The CI model is described as a method for explaining how humans comprehend written or spoken text. Kintsch's model attempts to explain how humans create their mental models of the world based on the current situation, current goals, and the huge amount of previously gained world knowledge that may or may not be applicable to the current task. It does this through the spreading of activation from nodes in a network that have been activated to varying degrees by their presence within the text, by their previous activations from context, and by the existence of world knowledge. Kintsch's aim is not just to model the result of human textual comprehension, but also to model *how* humans comprehend that text. In addition to the use of spreading activation to understand a situation, the model also displays concept priming in much the same way, as will be shown using neural schema mechanism.

The process for understanding a given line of text in the CI model is a multi-step procedure that results in a "mental model" of the text along with the creation of new knowledge of the world based on the text. First, a network of propositions is created based on a set of weak rules. These rules dictate how a set of proposition nodes is to be created and connected. Currently, there is no automated process by which these rules can be applied and, therefore, the initial network must be constructed by hand. The resulting network represents all of the system's previous knowledge including the constraints between propositions and the activation of propositions indicative of the current state of working memory. In this case, working memory would include the propositions in the text that is currently being processed and the recent context if available.

Once this network is created, a standard method of spreading activation is used to transform an initial set of activations of proposition nodes into a new set of activations that represent the understood importance of those propositions based on the text. Prior knowledge is intended to be part of the original network allowing for the integration of new information at the same time that the current situation model is being constructed.

One of the main points of this theory, and a primary issue in schema mechanism, is the integration of concepts from working and long-term memory (LTM) that are relevant to the situation at hand. For Kintsch, working memory (WM) is synonymous with the focus of attention and, consequently, the contents of consciousness. Long-term memory, on the other hand is, essentially, that portion of the proposition network that is not currently in working memory. One other distinction is made when referring to memory: Long-term working memory (LT-WM). This is a subset of long-term memory that has strong and, presumably, numerous links with the propositions in working

memory. Kintsch actually calls this system of links "retrieval structures." There is no distinct delineation between LT-WM and simple LTM except that LT-WM tends to be made up of propositions of relevance to the contents of WM that the individual has had a great deal of experience with. Therefore, LT-WM increases a person's ability to integrate concepts related to this area and makes memory recall more likely to be successful. Even though this distinction between LTM and LT-WM is important from a psychological perspective, it does not perform any special function in the implementation of the model.

There is a special role, however, for working memory. Processing of text occurs one word and one sentence at a time. At the end of a sentence that has just been processed, WM is, most likely, filled and must be cleared in order to allow for the following sentence to be processed. Therefore, after all of the propositions in the sentence have been integrated into the network, the WM is reduced to only a small number of key propositions dictated by the focus of attention. The key propositions left in WM serve as relevant context for subsequent sentences. While the exact number of propositions carried over and the method for determining them is not dictated by the model, it is generally the single proposition with the highest activation. Those propositions removed from WM are incorporated into LTM for possible future integration and assistance in understanding upcoming sentences. At the end of processing a sentence, the activations of the propositions in WM form the mental model of the meaning of that sentence. The proposition with the highest activation represents the major idea of the sentence. This sentence-by-sentence cycle is repeated for an entire

text allowing for a thread of the main idea to be maintained and integrating the knowledge gleaned from the passage into long-term memory.

An argument can be raised against this model from those in the AI community who would say that Kintsch has largely ignored the question of perceiving the propositions in the first place. While he does postulate the existence of a large and non-trivial set of production-like rules for the creation of these propositions, he makes little effort to enumerate what those rules might be, relying, instead, on humans to provide those propositions. Aside from this reliance, the theoretical use of symbol-based production rules leaves the model vulnerable to the symbol-grounding problem. The resulting criticism being that since the model deals only with the statistical co-occurrence of symbols that, rather than creating a mental model of the true meaning of a sentence, the system merely creates a statistical distribution of proposition co-occurrences. This is not really an argument against the model's correctness with regards to the process used, only to the completeness or applicability to the resulting output. In other words, one could say that the method of construction and integration does, indeed, model human cognition along these lines, but that it does not result in useful output because the input is flawed or incomplete.

Even so, this model should not be dismissed offhand; a couple of rebuttals could be made. For one, symbol-grounding issues do not discount the theory; they simply require better explanations of proposition acquisition that take this issue into consideration. Kintsch believes that the same method implemented to accomplish text understanding could be used to process visual or other sensory data. The real issue arises not out of what the mechanism does with the symbols that it uses, but from where those

symbols derive. In the text only version of the CI model, the symbols (words) and their organization are created by humans; there is no opportunity for the mechanism to ground those symbols is meaning except as it can be derived through the statistical analysis of their co-occurrence. The use of symbols as sensory features is not the problem here, it is the fact that the mental representation that the model produces, the output, is described as conveying the deeper human level understanding instead of the shallower word co-occurrence. On the other hand, if the mechanism is using its own internal processes to create its own world knowledge and somehow links that knowledge to a symbol, then the symbol grounding problem goes away. Kintsch argues that his model accomplishes this although this version has not yet been implemented. Second, a similar system that also suffers from this same shortfall, Latent Semantic Analysis (Landour & Dumais, 1997; Landour, Foltz, & Laham, 1998), has been shown to perform as well as humans when carrying out the task of grading student written essays (Landour et al., 1998). Latent Semantic Analysis is a method for taking a large corpus of text, say an encyclopedia, and creating a matrix representation of the meaning of all the salient words. This is done by first creating a word-by-word matrix noting all of the times that each word appeared in the same sentence with the other words. Then a statistical algorithm is used to reduce this matrix into a compressed form, which contains the semantics of each word as a vector in the new matrix. The meaning of a sentence or passage is found by adding the vectors of all of its words to form a single vector, and the similarity in meaning of any two passages is simply the cosine of the difference between their vectors. This model, even more than the CI model, displays a distinct lack of symbol grounding, and yet the system performs quite well along side humans. The argument, in light of this data, is that

there must be a strong correlation between this statistical data and the mental models that humans actually create. If symbol grounding is so important, then why does this data say otherwise?

The intention here is not to argue for or against either side; this may well be one of those instances when both aspects are correct. It seems quite evident that the CI model may lack adequate symbol grounding and, therefore, may be limited in the extent of human understanding that it can capture. On the other hand, the evidence suggests that in the realm of text comprehension, at least, this limitation is minimal. It is likely that the shortfalls of this system would become more evident if it were coupled with a system to produce real-world action from the text. While the model could, no doubt, form a plausible representation of the text, there is no way for that representation to include a series of actions and sensory expectations that would be part of a human's understanding of that same text.

Of course, to be fair, this was never the purpose of the CI model. This is, however, the purpose of neural schemas. It is hoped that, ultimately, the differences in these two domains will be shown to be merely a difference in abstraction level and that a single model can bridge the gap between sensory/motor and high-level text comprehension. The neural schema mechanism has taken a first tentative step toward this goal, but will probably not complete the journey in its current form. Even though Kintsch's model and neural schemas seem to be focused on different areas of human cognition, there is a good deal of commonality in the approach and methodology used to accomplish their respective tasks. Both the CI model and neural schemas rely primarily on structures described at a higher level of abstraction than would be present in a

standard connectionist system while still utilizing the spreading of activation to represent the current state of the system including its understanding of the present state of the environment. This places both squarely in the realm of classic/connectionist hybrids that meld the functions of classic symbolic AI with the dynamic aspects of neural networks. Another interesting point of similarity is the importance of "consciousness" in determining the model's understanding of the situation. While there is no explicit working memory in neural schemas, the attention mechanism serves a very similar purpose. Perhaps it would be beneficial to incorporate some of the memory features of the CI model into the neural schema mechanism although there are no plans for such at present.

8 Neural schemas

The new implementation of schema mechanism is called a neural schema mechanism because of its translation of the original structures into node and link elements[5]. This section will detail the neural schema mechanism and how its structures interact to achieve robust learning and goal directed action.

In general, one can think of the mechanism as a network of nodes connected via links (see Figure 8.5). Each node has its own activation, which it can spread to other nodes through the links. The links perform a function on the activation sent through them from their input node and deliver the resulting activation to their output node. With the exception of some rules that dictate which links are allowed to transmit activation, this part of the mechanism has a distinct connectionist flavor. Abilities are added to the nodes and links of the network that allow them to keep track of the necessary statistical data for the schema mechanism. In addition, nodes have the ability to check the statistical information in order to determine if a new node needs to be created and have the ability to create the new network elements.

The new mechanism uses the lingering and spreading activation features of connectionist systems (as described above) as a major component of action selection. It may seem as though such changes are a drastic divergence from the original mechanism and may not perform equivalently, but in actuality this is not the case. All of the abilities of the original mechanism have been preserved while extending some features and marginally reducing the computing resources needed. For example, the act of selecting a

[5] Drescher actually views the schema mechanism as a connectionist network but only to the degree that the schema slots can be interpreted as links; there is no spreading of activation.

schema for activation is now done solely on the basis of activation levels and applicability as opposed to a set of rules that gauge the relationships between reliability, result-action correlation, cost of activation, and applicability. This modification changes the way that the mechanism acts in a subtle but important way related to how activation is spread through the network. Another important change that has been made to the system involves the redefining of how goal pursuit is accomplished. In neural schemas, the idea of maintaining a cost for the activation of each schema is replaced by the notion of spreading desirability initiated from a node in the network representing a goal state. This method preserves the opportunism of the original system implemented using action controllers while removing the need for maintaining these large structures. The use of broadcasting of goals has also been expanded to more closely approximate a psychological theory of consciousness (Baars, 1988, 1997). In addition to providing a stronger psychological foundation, this allows the mechanism to discover novel solutions to otherwise insurmountable situations.

Throughout the course of this dissertation, examples and illustrations from a sample environment will be used where appropriate. The environment chosen for these examples and used for the experimentation of the mechanism is called Wumpus World (Russell & Norvig, 1995) reprinted for your convenience in Figure 8.2. Figure 8.1 shows an example of a neural schema network for the Wumpus World in its initial state. The names of the nodes at this point accurately reflect their purpose. To summarize, however, the agent, in this case the hunter, has the following percepts if he is within one square of a given object: the glitter of gold, the breeze from a pit, and the stench of the wumpus. The hunter can also sense the following percepts if he is within the same

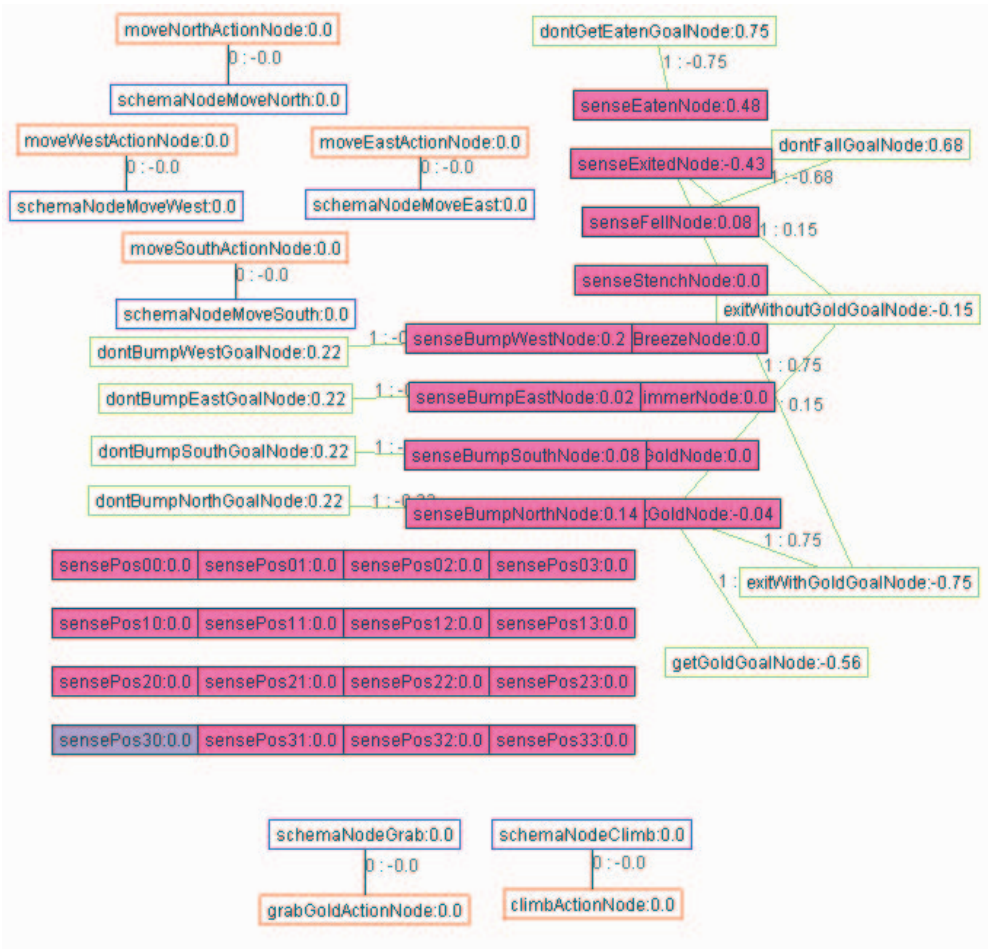

Figure 8.1: Initial state of a neural schema network for the Wumpus World environment.

square as a given item: the gold, being eaten by the wumpus, and falling in a pit. The agent also senses his position, whether his is carrying the gold, if he bumps into a wall (directionally significant), and if he has exited the cave (grid). The hunter has the ability to move in any of the four directions (north, south, east or west), grab for the gold, and climb out of the cave. The original Wumpus World game also allowed the hunter to shoot the wumpus with an arrow; for simplicity sake, this feature was not imparted to the neural schema version of the hunter. The objective for the agent is to locate the gold and find his way out without being eaten by the wumpus or falling into a pit. To reflect these

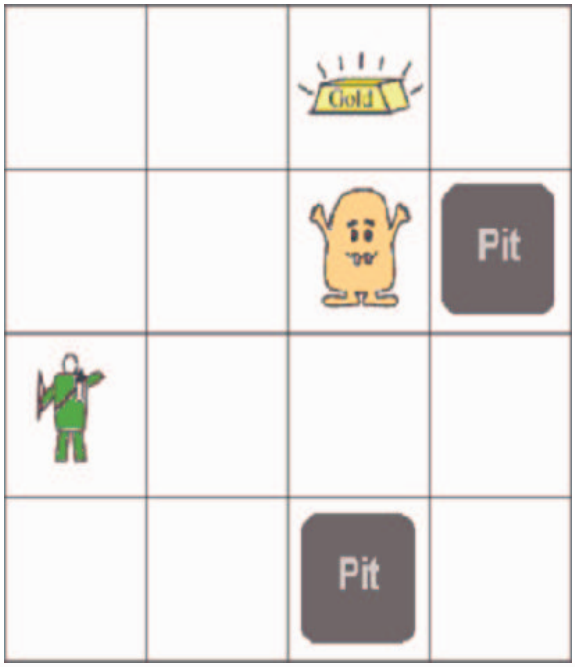

Figure 8.2: Wumpus World (Russell & Norvig, 1995)

overall objectives, goal nodes attach to the appropriate sensory item nodes and denote the relative like or dislike of having that item on or off. In addition to the goals directly related to the objectives, the hunter also has a mild dislike for running into walls (the edges of the grid).

8.1 Links

In the neural schema mechanism, links perform all the same functions as you would expect links to perform in a connectionist network. Even though each link is of a specific type, the workings of every link is, for the most part, the same; the type labels are used by nodes in the system to determine if it needs to deal with that link in some special way. The link types used in the current implementation are *context*, *result*, *goal*, *action*, *host*, and *none*. Table 8.1 lists the different types of links and gives a brief description of their

key functions. Parts of the descriptions may not make a lot of sense at this point, but the table should come in handy as a quick reference for future sections.

Table 8.1: Link types

Link Type	Description	Primary Purpose
Context	Connects an input Item node with an output Schema node	Marks an Item node in a given state (*on* or *off*) as having a correlation with the successful execution of the Schema node
Result	Connects an input Schema node with an output Item node	Marks that the Item node transitions to a given state (*on* or *off*) when the Schema node is executed
Action	Connects an input Schema node with its one associated Action node or connects a learned input Action node to its learned output Goal node	Provides a path through which an Action node can send information back to its calling Schema node(s) or provides a path through which a learned Goal node can send information back to its calling learned Action node
Goal	Connects an input Item node with an output Goal node	Marks the state of the Item node that the Goal node is trying to achieve or avoid and provides a path through which desirability can be sent from the Goal node
Host	Connects an input learned Item node with an output Schema node	Mark those Schema nodes from which the learned Item node derives its state
None	Connects an input Item node to an output Schema node or connects an input Schema node to an output Item node	Marks Item nodes that are potential contexts or results of the Schema node's execution and maintains the statistics necessary to determine whether appropriate spin-offs of the Schema node should occur

With the exception of action and goal link types, a link must connect an item node to a schema node.

Figure 8.3: Links generally connect item nodes with schema nodes

A link with an item node as its input will have a schema node as its output, and is a context type link or is considered a candidate to become one. Conversely, a link with a schema node as its input will have an item node as its output and is either a result type link or is considered a candidate to become one, or is a host link that connects a learned item node to its host schema node. An action type link connects a schema node to its own action or connects a composite action to its goal node. Goal type links connect goal nodes to the item nodes that make up its goal set. Links that do not have a type (technically of type *none*) are either part of a schema node's extended contexts or extended results. These amass statistical information but do not participate in passing of activation or desirability. Figure 8.5 shows the early stages of a schema network running in the example environment described previously. The light colored links shown going from schema nodes to sense (item) nodes are of type *none*. The dark links from schema nodes to action nodes are of type *action*.

The statistics maintained by the links are the primary source for learning new network nodes. The first statistic of note is the *relevance* of the link, more technically known as the ratio of the positive and negative transition correlations. A *transition correlation* is the likelihood that a given item node will change its state to *on* (positive) or *off* (negative) following the schema node's activation. If either the positive or the negative-transition correlation for an item node becomes sufficiently greater than the

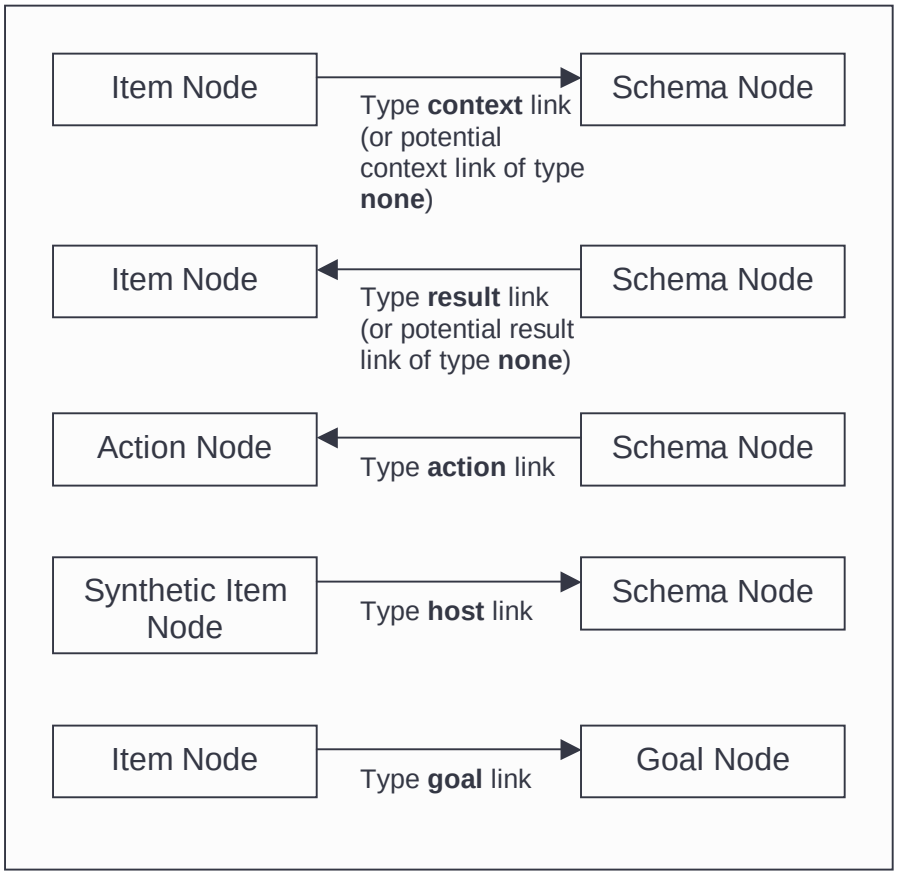

Figure 8.4: Possible link configurations

other, then a new schema node is created with that item node linked via a link with its inclusion state matching the greater transition correlation. The relevance calculation is slightly different for possible result links (from schema node to item node) and possible context links (from item node to schema node). For possible context links, the correlations do not take into account the transition of states but, rather, the state that the item node was in when the schema node's action was executed. In other words, the relevance of a link measures how important the item node is to the successful completion of the schema node's action either from a context or result stand point.

A link's basic duty is to pass activation from its input node to its output node after that activation has been modified by its weight. In this respect, neural schema

mechanism links are quite similar to a link in any standard connectionist network. However, this is where the similarity ends; almost every other aspect is unique to neural schemas. The weight of a link, for instance, is only meaningful for *result* and *context* type links, since they are the only types that transmit activation or desirability. A result link's weight is the product of its relevance and its input schema node's reliability value. For a context link, the weight is the product of its output schema node's correlation value and its output schema node's reliability. The weight can, however, be temporarily modified by the desirability flowing backwards through the link. Desirability will be explained in more detail later, but for now one could think of it as analogous to the conductivity in an electric circuit. It effects the activation flowing through the system by modifying the strength of the weights on the links. Desirability will also be used to learn the correct values for any intermediate goal states.

Links also serve as the contexts, extended-contexts, results, and extended-results for schema nodes. Each link maintains statistics on the relevance between the input node and output node as well as positive and negative transition correlations. Relevance, as defined earlier, relates the probability that the output schema node will succeed when the input item node's state is *on* to the probability that the output schema node will succeed when the input item node's state is *off*. This value is used by the output schema node to decide when a new schema node needs to be created with this link added to its context set. Likewise, the ratio of the positive and negative transition correlations is used by the input schema node to determine the need to create a new schema node with this link added to its result set.

At first, a neural schema network has almost no links. In fact, at the time of a

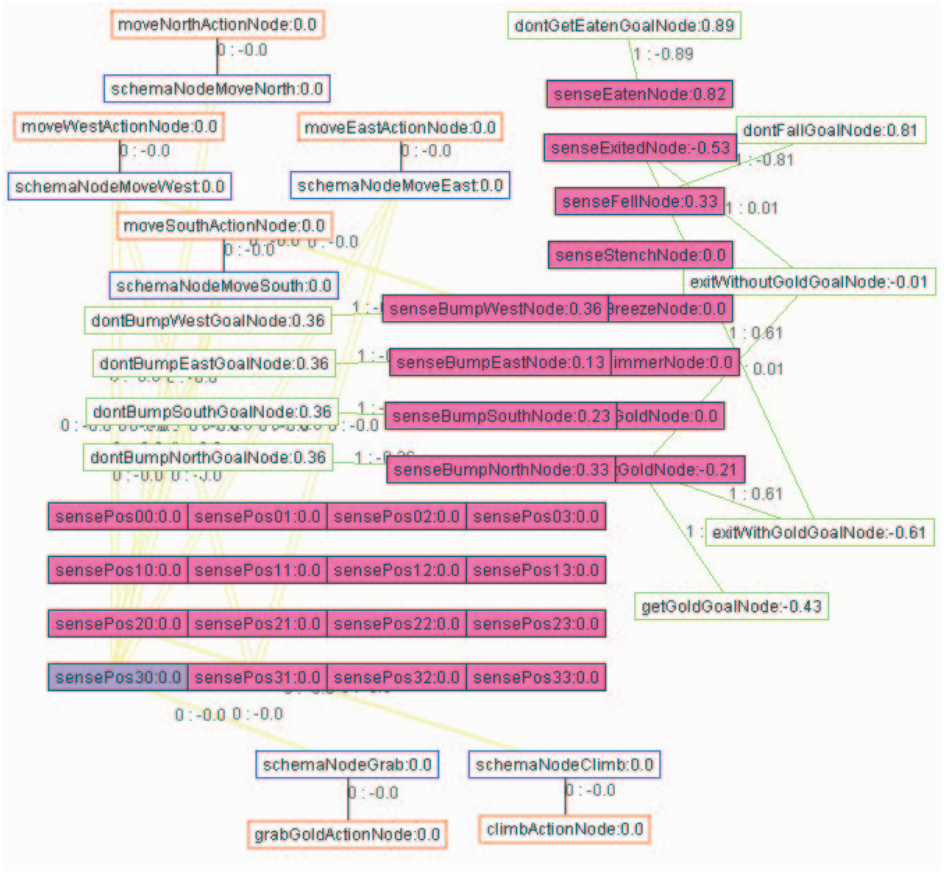

Figure 8.5: Early stages of a neural schema network. Light colored links are potential context or result links.

network's initialization, the only links that exist are action links from primitive action nodes to their corresponding bare schema nodes or, possibly, from primitive goal nodes to their goal set's item nodes. Figure 8.1, introduced earlier, shows the initial state of the neural schema network before any processing has occurred. An attention mechanism will be discussed later that creates the initial links between nodes, such as those shown in Figure 8.5.

8.2 Nodes

The node, as with most connectionist systems, is one of the two main components, links being the other. A node in the neural schema mechanism can come in a variety of types

that act differently based on which type it is. However, one commonality for all of the nodes in the system is that they have a state that is in correlation with their activation.

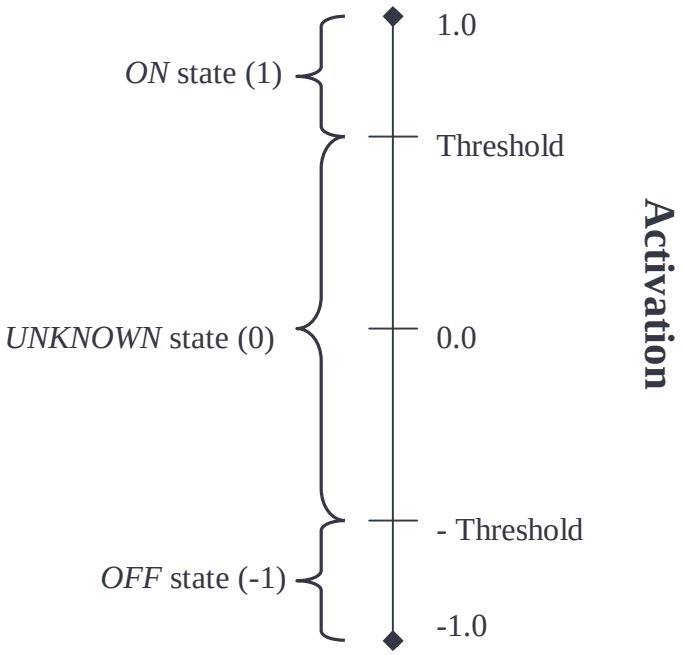

Figure 8.6: Activation to state mapping in nodes

Depending on the type of node, the activation may be adjusted to match the node's state, or the state may be determined by the node's activation. Most nodes have the ability to pass activation and desirability through the network, although there are limits to this ability based on the types of links that the activation or desirability could flow down. Activation may be propagated from a node when the node's activation goes over some threshold or under some negative threshold. It is said, in these cases that the node has "fired." Firing a node, however, does not mean, in the case of schema nodes, that its action is executed.

8.2.1 Schema nodes

The most complicated type of node is the *schema node*. These nodes represent the center of the explicit knowledge structures of the network. A schema node, as you might suspect, correlates strongly with the schema of the original mechanism. The main difference lies in the way that the schema node keeps track of its contexts and results. Specifically, the schema node has a list of input links, which hold information about the node's context (and potential context) and a list of output links, which hold information about the node's result (and potential result). An additional output link points to the schema node's action. An individual schema node embodies the knowledge that the item nodes pointed to by the result links will transition to a given pattern when the action node is activated under the conditions denoted by the item nodes pointed to by the context links. Put another way, the schema node states that a certain result is more likely to occur than not when the schema node's action is executed within a given context. Note that the schema node's existence does not imply that the result is likely, only that it is more likely than not when the action occurs within the context.

To measure how reliable a schema node is the node keeps a running tally of successful vs. unsuccessful activations and calculates their probabilities. The trick comes in determining when a schema node has been activated. As you will recall, a schema node can either be explicitly activated, meaning that it was specifically chosen for activation, or it can be implicitly activated, meaning that another schema node which shares the same action node was chosen (Figure 8.7). What's more, some of the statistics necessary (i.e., reliability) do not care whether the activation was explicit or implicit, while others (i.e., correlation) do need to know the difference. While the *reliability*

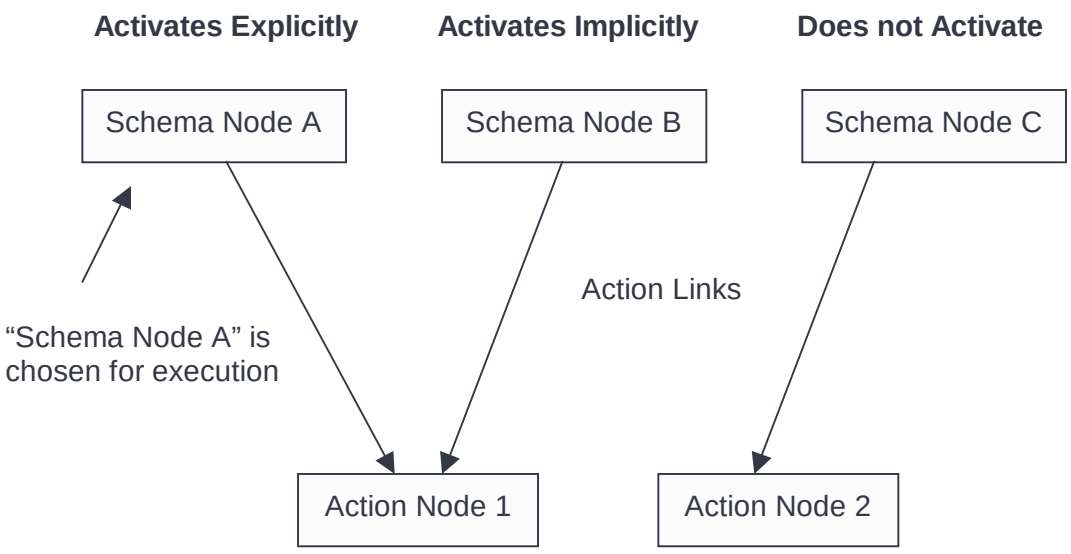

Figure 8.7: Explicit vs. Implicit activation

measures how essential this particular schema node is in bringing about the designated result when its context holds, the *correlation* of a schema node measures the relationship between the action and the results. For this, the node needs to know not only when its result occurs after its action's activation, but also when it was not activated and the result occurred anyway. This is handled by letting the action node propagate its completion state (either *successful* or *failed*) down its input links to the schema nodes that could have activated it. Each schema node notes when it has been specifically chosen for activation. Therefore, the node that actually instigated the action knows who it is and can adjust its statistics accordingly; others can correctly assume implicit activation. The schema node is also watching it's result item nodes in order to note when a result transitioned without activation of either kind.

The reliability of the schema node is calculated at the point when its action propagates a new state. By keeping up with the number of times that this propagation

occurs (number of activations) and the co-occurrence of these activations with a successful result state, the schema node can calculate its general reliability. At the same time, the node is also keeping track of when activation follows within a short time of another successful activation. These events are included in the statistics for local reliability. *Local reliability* is based on the notion that things in the world tend to stay put, at least for a while. Therefore, a schema node can be generally unreliable but still locally reliable meaning that if the schema node has just been successfully activated then a repeated activation will likely also be successful.

A more difficult statistic to maintain is a schema node's correlation. Once again, correlation is defined as the ratio of the probability that the schema node's result will obtain when activated to the probability that the result will obtain when not activated. Here, since activation implies applicability, the node needs to know not only when it was activated, but also when it was *not* activated when it could have been. For this reason, all schema nodes that were not explicitly chosen for activation, but whose results transition into an *on* state (the result set obtains), must check whether their actions are currently running. If they did not receive a completed action state, then their action must either be *off* or still in the process of running. Once the schema node knows the state of its action, it can correctly calculate or defer calculation of the probability that its result state will obtain when it is applicable but not activated. This would be the case even for situations where the agent did not instigate, through direct actions, the transition into the result state. For values where the negative-transition correlation probability is non-zero, the function (Equation 8.1) results in mapping the raw correlation ratio to a value in [-1, 1]. Any raw value above one means that the positive-transition correlation probability is

$$\frac{2}{1+e^{\left(\frac{-x}{5}\right)+5}}+1$$

Where x is:

$$reliability \Big/ probability\_ratio$$

Equation 8.1: Schema node correlation function

higher than its negative-transition counterpart and will be mapped to a positive value correlation which increases as the difference between the two probabilities increase. A raw ratio value between 0 and 1 means that the negative-transition correlation probability is higher and will, similarly, be mapped to a negative correlation value. In cases where the negative-transition correlation probability is zero, the correlation is equal to the reliability.

All of the statistics maintained by schema nodes in the mechanism serve one purpose: to aid in determining which action to perform at any given moment. Action selection is conducted based on the product of a schema node's activation and correlation. As we will see, many other factors come into play in determining a node's activation, but the final selection is performed solely based on these two features. The action selection process will be described in greater detail once all of the required elements have been explained.

The other main function of schema nodes is to create any new nodes that the mechanism has deemed appropriate. Even though it is, technically, the mechanism as a whole that gives the go-ahead to the creation of new nodes, it is the individual schema

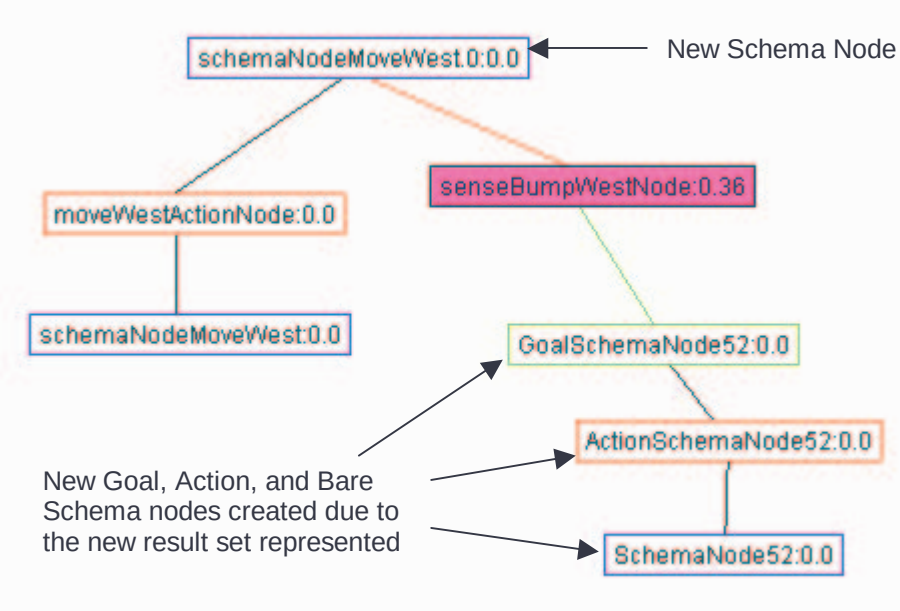

Figure 8.8: A portion of a neural schema network after a result spin-off. Type *none* links have been removed for clarity.

node that decides whether its local conditions warrant beginning the creation process. Up to this point, however, we have not talked about the initial state of the original schema nodes as seen in Figure 8.1. A schema node's most basic state is when it does not have any result or context links and has only its one action link; these nodes are called *"bare" schema nodes*. It is from this original bare state that a schema node can spawn new schema nodes with additional context or result items and can create synthetic (learned) item nodes. In instances where the result list of a spin-off schema node is new to the system (no other schema node has this same result set), a schema node can also create a new composite action with a new bare schema node to activate that action node. Figure 8.8 shows a portion of the network for our example agent at the point where the hunter has discovered that moving west ("`moveWestActionNode`") can result in his sensing a bump to the west ("`senseBumpWestNode`"). This learning manifests itself in the mechanism as a new schema node that has been created with

"moveWestActionNode" as its action and "senseBumpWestNode" as its result. Also, since this is a new result set for the network, a new action node is created along with a corresponding goal node and bare schema node that can be seen to the bottom right of the figure. The goal node is connected to "senseBumpWestNode" denoting the new result set as the goal of "ActionSchemaNode52". A key difference in the two schema mechanisms lies in the way that composite actions are performed and managed, although the details of this must be differed to the section on action nodes.

As with any other type of node, a schema node can pass its activation when it fires. This is distinctly different from the mechanism choosing a schema node to activate. Remember that activating a schema means that the schema node is applicable and that the mechanism has chosen to perform its action. Firing, on the other hand, occurs whenever a schema node is applicable and its activation goes over a threshold or under a negative threshold. When this happens, it sends activation through its result output links.

8.2.2 Item nodes

An item node corresponds to some state in the environment. Primitive item nodes, like primitive items in the original mechanism, are connected directly to some sensory apparatus. Unlike schema nodes that must rely on other parts of the mechanism to provide them with their activation, primitive item nodes set their own activation based on the values input from the sensory apparatus. For switch-like senses, such as a bump sensor, an item node's activation might be set to 1 if the bumper is depressed, or –1 if the bumper is not depressed. Other senses, such as proximity sensors, might map an item node's activation to a continuous value between –1 and 1. It is important that each primitive item node's range of activation be in [-1, 1]. A threshold function determines if

an item node's state is *on*, *off*, or *unknown* based on the activation of the node. If a node's activation range were only in [0, 1], for example, then the threshold function would set the node's state to *on* or *unknown* but never *off*. For learning purposes, such a situation would yield incorrect or, at best, unreliable information about the node's relation to schemas and actions. In our Wumpus World environment, the state of each of the hunter's senses is boolean, rendered in the network as [1] for *on*, and [-1] for *off*.

When a schema node is not generally reliable but its local reliability is high, then there is the possibility of learning a new item node. Unlike primitive item nodes, learned item nodes, what Drescher calls "synthetic" items, are not directly connected to any sensory devices. Instead, they are connected via a "host" link to the schema node that spawned them and any of its subsequent context spin-offs. Learned items nodes are an important part of the discovery process within the mechanism. They establish the counter-factual assertion that a given action *would* produce a given result if that action were taken under a specific set of circumstances. An example might help to show the importance of this element. Suppose that an agent is in a dark room. The agent has noticed that, sometimes, when it takes a step forward, that it runs into something. In general, the act of taking a step forward does not usually result in the agent running into an object. However, if it has just tried to step forward and stubbed its toe, and then repeats the same action, it will stub its toe again. A learned item is created with this locally reliable but generally unreliable schema as its host. The schema is said to reify the item. In other words, the schema designates the conditions, through its context set, under which the item's state is *on* (or *off*) even though the agent doesn't initially know what those conditions are. Further context spin-offs of the schema node also become

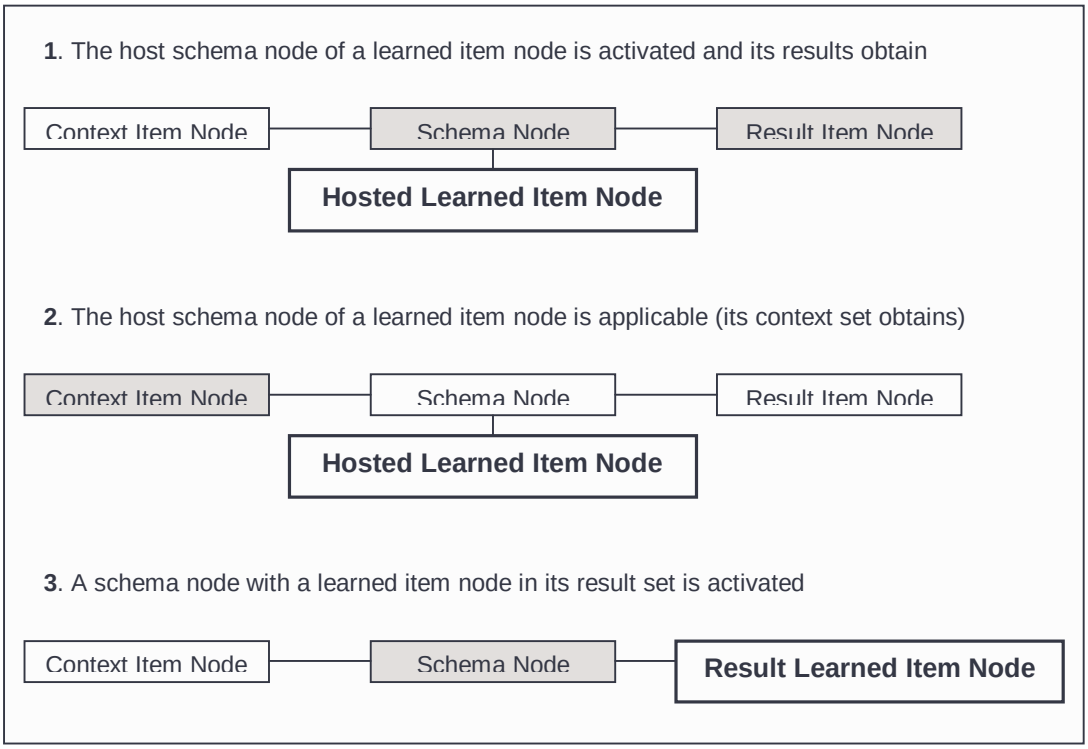

Figure 8.9: The three situations when a learned item node's state is turned to *on*

hosts to the learned item node and will refine its ability to note the state of this node. The mechanism can treat that item as an object such that it does not need to actually move forward while at the special location, but can create chains of schema nodes that move the agent around the object without running into it. Even though this is a simple example, it demonstrates how learned item formation can change the behavior of the system in a very important way. From this point, schema nodes can be created whose contexts or results contain the newly created item node.

Unlike primitive item nodes that rely on sensory apparatus to determine their activation and state, learned item nodes must find some other means to determine their state. There are three methods that a learned item uses to maintain its correct state. The most obvious is the successful execution of any of its host schema nodes. Since the

learned item node defines the conditions under which the host schema nodes will be successful, a successful execution of one of those nodes is the clearest indicator of the learned item's state. Secondly, if any of the learned item node's host schema nodes are applicable (their context set is satisfied) then the item node should be turned on. All of the host schema nodes will have the same action and result; therefore, context spin-offs serve to more accurately specify the conditions in which the result follows the action. Having any of a learned item node's host schema nodes have their context satisfied is equivalent to stating that the conditions are correct for the result of the schema node's action to obtain and, hence, the learned item node is turned on. The final way that a learned item maintains its state is through the prediction of that state by being included in the result set of some schema node.

The learned item reifies its hosting schema nodes by allowing the mechanism to treat the arbitrarily abstract conditions laid out by the host's context as a real and palpable entity. In the case where one of the learned item node's host schema node's context is satisfied, and, therefore, the item node's state is *on*, the mechanism does not have to perform the action to know (or at least guess) that the result would occur. Under these circumstances the mechanism could learn to perform actions based on the existence of the conditions represented by the learned item node without having direct sensory manifestation of the result. The mechanism is thereby treating the learned item node as though it represented the definition of the existence of the object. For instance, let's suppose that there is a learned item node in the Wumpus World that is originally hosted by a schema node whose action is to move west and whose result is to feel a bump. A human observer might say that the learned item node represents the fact that there is a

wall to the west. Subsequent context spin-offs of the original host node refine the agent's ability to recognize when it is in a situation where there is a wall to its left. These contexts might be positional in nature, such as when the hunter is at position 2, 0 (third row, zeroth column). Because of the second rule for when a learned item node is activated (its state is set to *on*), which states that whenever the context set of any of its host schema nodes are satisfied, the learned item node is activated, this "wall to the west" learned item node will be *on*. Since the learned item node is *on* even though the action of moving west was not taken, the mechanism can learn to act appropriately with regard to a wall being to the west even without having direct sensory confirmation, the feeling of the bump, that there is a wall to the west. Over time, with further context spin-offs, the mechanism will be able to accurately turn *on* the "wall to the west" learned item node when there is, actually, a wall to the agent's west. By adding this learned item node to the context and result sets of new schema nodes, the mechanism learns about the abstract notion of a wall being to the west.

It is important, however, not to confuse reification with definition. In our "wall to the west" example, a learned item node might be created that reifies the schema node that states that (after one or more context spin-offs) at a specific location the action of moving west results in a feeling of a bump. One might erroneously conclude that the learned item defines the existence of the wall since the mechanism treats that item as though it represents the object. Instead, what the item represents is the conditions under which the existence of that object, the wall, could be confirmed. Over time, the agent can create context spin-offs of host schema nodes that continue to more closely specify the reification conditions, perhaps to a degree that is near infallible. Even so, the agent will

never be able to construct a *definition* of the wall, only closer approximations of confirmation conditions. The importance of this point comes out when one examines arguments against artificial intelligence being able to construct truly novel concepts (Fodor, 1975). The argument, put much too simply, states that an artificial system can only create reorganizations and abstractions of its inputs and, therefore, can never create concepts for things that are incomputable or for which only incomplete data is available. The logic of the argument is absolutely correct if one assumes that a definition is the infallible concept of something. The idea of learned items in the schema mechanism gets around this argument by not trying to define objects or concepts. A counter argument, of which the schema mechanism is an example, is that definitions of the type called for in the argument do not exist even in humans. For an interesting exercise to demonstrate this point, get a small group of people together and try to come up with an infallible definition for a chair. You will, most likely, start by listing the properties of a chair and will quickly discover that for every property there is a counter example. You may then resort to defining the chair by what one does with it, namely sitting. But one can sit on a lawn, or a rock, or a table. Finally, you may come up with a sort of fuzzy definition saying that everything has some degree of chairness, even a lawn is a chair if one sits on it. Even for something as basic as a chair, strict definition is difficult if not impossible. There are, of course, strict definitions for such things as mathematical concepts, a square, for example. Even so, it is quite possible that humans maintain only an approximation of the meaning of the elements that make up even these definitions. The schema mechanism learns, not fuzzy sets, but fuzzy conditions that allow the agent to interact appropriately in every way with that object or concept.

8.2.3 Action nodes

Action nodes perform the function of actions in the network. Although the action nodes are not the actions themselves within the implementation, this point is a technicality and not important to how the mechanism functions. Similar to item nodes, there are primitive and learned action nodes. For a primitive action node, an action controller is provided that directly connects to some actuator. These would be such things as particular muscle movements or turning on an engine.

Learned actions are created whenever a schema node creates a new schema node with a novel result. Here, "novel" is used to mean a result set that is not shared by any other schema node. When this occurs, the creating schema node carries out three additional functions, creating a new action node, creating a new goal node, and creating the bare schema node that allows the system to learn about the results and contexts of executing the new action. An example can be seen in Figure 8.8 where "`SchemaNode52`," "`ActionSchemaNode52`," and "`GoalSchemaNode52`" have been created in response to the new result shown in "`schemaNodeMoveWest.0`". The action controllers for learned actions in the neural schema mechanism are very simple and perform the same task for every composite action. Essentially, their only job is to monitor the state of their goal node and to activate the goal node when that action is executed. This activity will make more sense once the details of goal nodes are explained. The purpose for this multi-layered approach to action execution is merely to allow a single action node class in the implementation to provide the interface for all possible actions. This feature, by itself, is functionally no different than Drescher's original mechanism, which created an action controller for each composite action. These

original action controllers, as previously discussed, were required to keep track of the proximity of every schema to their goal state. In the new implementation, this interface class and the addition of a goal node replace the action controller as defined in the original mechanism.

8.2.4 Goal nodes

Goal nodes, essentially, have one main purpose; that is to provide value judgments to specific states of the environment or to individual item nodes; they are the source of motivation in the network. The original schema mechanism accomplished this task through the use of primitive, instrumental, and delegated importance values associated with items. One difference in the new implementation is that goals can be primitive[6]. In the original schema mechanism, a goal was always associated with a composite action; this is not the case with the neural schema mechanism. A primitive goal node is analogous to basic drives; its purpose being to influence the mechanism in such a way as to bring about its goal state, however that might be accomplished. Whether the goal node is primitive or learned, the method for bringing about its goal state is the same, namely the spreading of desirability through the network.

A primitive goal node in the new mechanism has an intrinsic primitive desirability value provided at design time. Instrumental value, on the other hand, is imparted through the propagation of that desirability at runtime. Delegated desirability value accrues in learned goal nodes. Remember that new goal nodes are created whenever a schema node is created whose result is new to the network. For this reason, there will be a goal node

[6] Drescher describes the possibility of moving the innate drive for discovery, which was built into the mechanism as a pseudo-randomly changing focus of schema selection, to an item that would register whether significant learning was going on. Goals, however, were only internal structures of composite actions.

associated with every unique set of item nodes that is referenced as a result of any schema node. The delegated desirability is calculated as follows. First, the maximum amount of desirability flowing through all applicable schema nodes is determined. This step measures the greatest benefit that the agent believes could be reached in one time step from the mechanism's current state. Each learned goal node then receives that value and averages it with previous values noting whether the goal node's state is currently *on* or *off*. For example, suppose a goal node is pointing to its single item node of "see a glimmer." This item node is *on* (the agent is seeing the glimmer) and the goal link's include state is *on* (meaning that the *on* state of the item node is the state that the goal node is concerned with). All of this means that the goal node is manifest or, in other words, is currently being satisfied. To calculate the delegated desirability value for this learned goal node on this cycle, the system first adds the current maximum desirability value that can be reached within a single action and adds this amount to the desirability values from all previous cycles when the goal node was also being satisfied. It then averages these values to come up with a new average desirability value for when the goal node is manifest; let's say that this comes out to an average of 0.8. The average for when the learned goal node is not manifest is left unchanged since that is not the case for this cycle; let's say this value is 0.2. The goal node's delegated desirability value is then calculated as the average maximum desirability when the goal is met (*on*) minus the average maximum desirability when the goal is not met (*off*). In other words, in our example, the 0.2 value for when the learned goal node is *off* is subtracted from the 0.8 value for when the learned goal node is *on* to come up with a new 0.6 delegated desirability value for this learned goal node.

Delegated desirability value in learned goal nodes is the method by which the mechanism learns the desirability or non-desirability of intermediate states or sub-goals. The item nodes and their states referenced by a goal node and its goal links make up a possible intermediate state along the path to one or more primitive goals. Determining the relative value of attaining (or avoiding) that state irregardless of the current drives of the system will allow the mechanism to anticipate states that may be useful in the future. The mechanism then generally attempts to maintain useful states or avoid detrimental states as long as there are no stronger conflicting goals that require otherwise. In other words, the system begins to pursue states for their own right without regard to the ultimate drives that they serve. This ability to pursue (or avoid) goals that are in service of some lower-level drive without reference to that drive is common in humans – even to the point of having mistaken priorities. The pursuit of money, for instance, can often become the singular focus of a person's life even though the money, itself, is not part of the individual's base drives. It is the things that can be purchased with the money that are desired, but the intermediate goal of acquiring money can subsume or obscure the underlying motivation. Even so, most humans seem to do quite well satisfying their basic drives. Although the neural schema mechanism has not produced behavior tendencies toward these human extremes, the theoretical possibility of this feature of goal nodes is intriguing.

In implementation, goal nodes perform two main functions, to compute a state function based on its goal links and to send desirability (positive or negative) backwards through the network when its action is executed. The goal links are approximately equivalent to a context. When all of the states of the item nodes pointed to by these links

obtain, then the goal node's state should be *on*, otherwise its state should be *off*. It might be possible to derive useful inferences and actions by letting the goal node's activation represent the degree to which its state obtains. Although in the current implementation this is not done, it would be a trivial task to modify the system to function in this way.

A goal node will periodically poll its goal links to determine its state, which is then passed on, in the case of learned goal nodes, to the action that has this as its goal. When the goal node's state turns *on*, this is noted by the goal node's action node, which will then propagate an action-completed signal to all of the schema nodes that might trigger it. Since obtaining the goal node's state is the defined purpose of the learned action node, having the goal node's state obtain without the explicit execution of the action node is equivalent to the implicit execution of the action. Propagating the action-completed signal whenever the goal state manifests lets the schema nodes connected to the action node learn about contexts and results of composite actions using both implicit and explicit activation just as they would with primitive actions.

8.3 Desirability, activation, and action selection

The ability for the neural schema mechanism to pursue multiple conflicting or complimentary goals is accomplished via the propagation of desirability through the network. Desirability begins in a goal node and can be positive or negative. We would say that a goal that is desired would propagate a positive desirability and one for which avoidance is preferred would propagate a negative desirability. Figure 8.10 shows a portion of the early state of a neural schema network for the Wumpus World environment. The colored in nodes in the figure are some of the primitive item nodes for the environment and the outlined nodes are the primitive goal nodes associated with

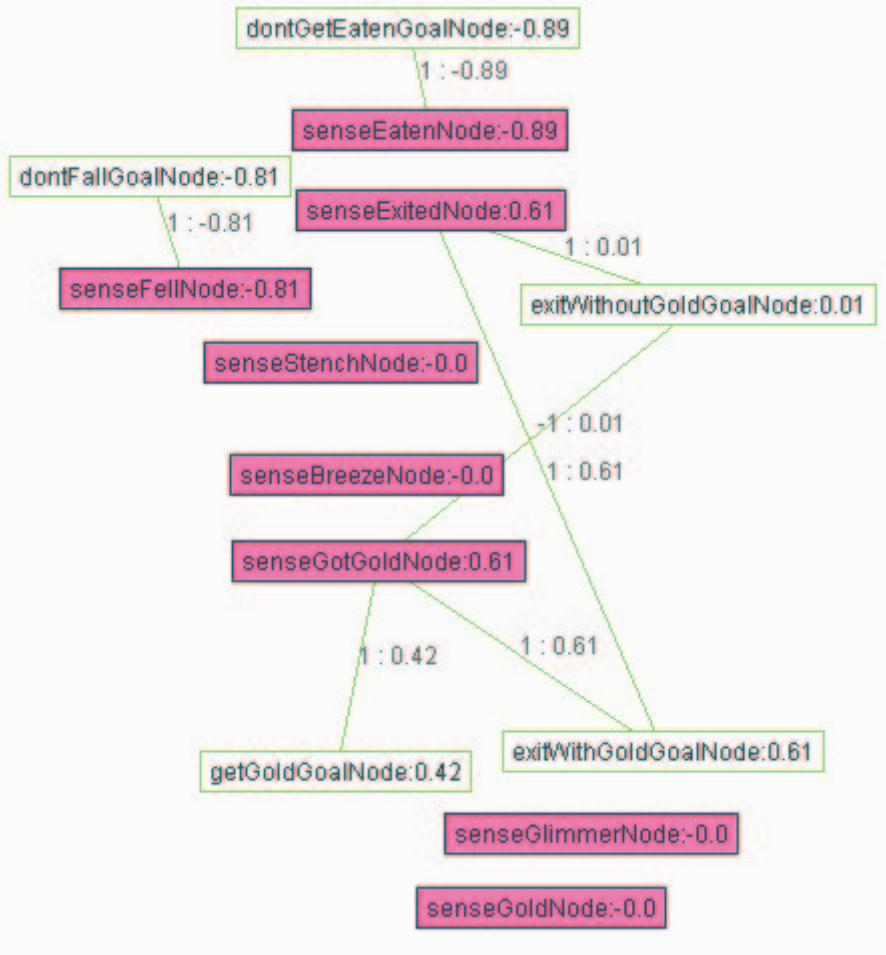

Figure 8.10: Some of the sense (item) nodes for the Wumpus World environment and their primitive goal nodes. The number at the end of the node names is the resistance at that node. The first number associated with a link is its include state and the second number is the resistance flowing across the link.

them. The numbers at the end of each node name displays the desirability of the node. Each link has two numbers labeling it; the first is the include state (either -1 or 1 in this case) while the second is the desirability flowing over that link. The desirability or undesirability of any given state of an item node with respect to a particular goal node is defined by the desirability of the goal node and the include state. In the figure, for example, the "`dontFallGoalNode`" has a desirability value of -0.81. This alone tells us that whatever state is represented by that goal node's goal links is a state that the agent

wishes to avoid (although there could be other goal nodes that want to achieve this same state). When we also look at the goal link connecting the "`dontFallGoalNode`" with the "`senseFellNode`" item node, which has a positive 1 as its include state indicating an *on* state, we can determine that the "`dontFallGoalNode`" is meant to avoid turning on the "`senseFellNode`" item node. Another element of Figure 8.10 that is important to note is the fact that not all primitive item nodes must be linked to some primitive goal node, "`senseStenchNode`" for instance. In the initial state of the network, there is nothing to tell the agent that the stench percept is one to be avoided; it is up to the agent to decide for itself over time if this sense is one to be sought or avoided.

Just as with activation, there are limits as to which links are permitted to transmit desirability. Item nodes can only send desirability through result links to schema nodes and schema nodes are only allowed to send desirability through context links to item nodes. This process maintains the path to the instigating goal node with stronger desirability values occurring closer to that goal.

Desirability does not actually change the activation of a node; instead it affects the amount of activation that passes through a link. The best way to explain this would be to say that the desirability in the system alters the contours of the network. One can think of the neural schema network without desirability as occupying a flat plain within a three-dimensional space. Activation is like water trying to progress from an existing state to a goal state with the weights of links providing some hills and valleys that hinder or assist the water in its journey. The network landscape at this point in our analogy represents a purely logical perspective of how to get from one state to another. When desirability is added, the contours of the network change to reflect a combination of logic

and desires. What once was a valley might now be a flat plain or a veritable gorge; in the same manner, a hill might become a mountain or a plateau.

Goal nodes, the source of desirability, are activated in one of two ways. If the goal node is attached to a composite action then the action node can activate the goal node. If, on the other hand, the goal node is primitive, it might have some direct attachment to a proprioceptive sensor that would register something internal to the agent such as hunger. In a circumstance such as this, the goal node would be attached to a primitive item node or nodes that represent the proprioceptive state. While this may seem redundant, it is not. The item nodes, themselves, only register a state of the environment and do not make judgments as to its desirability or undesirability. Goal nodes, however, have the explicit purpose of "deciding" which environmental states should be sought and which should be avoided.

For goal nodes in service of composite (learned) actions, desirability is only propagated for a short period of time (based on the expected duration of the action) unless the action is repeatedly selected. Primitive goals will continuously spread desirability based on the current state of the system. This allows a primitive goal, such as hunger, to be pursued to varying degrees at all times. When the goal node's need is high, the agent may pursue it single-mindedly; however, once such a goal has been satisfied, it would be reduced to a level that has little to no effect on the behavior of the system.

While desirability begins at the item nodes pointed to by active goal nodes, activation has its source in item nodes that represent the current state of the environment. Activation flows in an opposite manner to desirability. An item node can pass activation through its context links to schema nodes, which can pass activation through their result

links to item nodes. As mentioned previously, activation can be transmitted from any node whose activation level exceeds a threshold or drops below a negative threshold. Since activation is spread through the system by numerous nodes, there must either be a limit to how high the total activation of a system can grow or the decay rate must be tuned appropriately to manage this level. Since the number of nodes in the network is always in flux, the total number of nodes in the network must be taken into account. In the neural schema mechanism, both methods are used at different times. The total amount of activation of a network is limited to half of the total number of nodes. If the total activation of the network is below this maximum, then only a decay rate is used to reduce the activation of a given node. However, if the total activation goes above the maximum, then the activation is normalized so that each node maintains its correct percentage of the whole but the total activation of the network remains equal to the maximum. The use of half of the total number of nodes as the maximum for network activation is not arbitrary but may need to be adjusted in future experiments. It may also be possible to allow this value to change during a run. For instance, lowering the value from 0.5 to 0.25 might promote more abstract thinking by forcing the system to highly activate fewer, hopefully more general, nodes.

Even though activation plays an important role in action selection, it is not the only element to consider. In addition to activation, a schema node's applicability is taken into account. Under normal circumstances, for a schema node to activate (perform its action) the node must be applicable and have the highest activation of all applicable schema nodes. The use of desirability increases the likelihood that the schema node selected for activation lies on some path to a goal state. In addition, the fact that the

desirability values become stronger the closer a node is to the goal state, means that the schema node chosen for activation is more likely to be one which is fewer steps away from achieving the goal. This method gives the neural schema mechanism the ability to be opportunistic while pursuing multiple goals.

8.4 "Consciousness" and attention

It was stated in the previous section that, under normal circumstances, a schema node must have the highest activation of all applicable schema nodes to be chosen for activation. The other way that a schema node could become active is through a response to a "conscious" broadcast. The term "conscious" is used here only to denote that the mechanism implements a portion of a psychological theory of consciousness[7]. It is neither our intent to suggest nor our belief that the mechanism displays true consciousness. That being said, it is our hope that, within the limited scope of the "conscious" mechanism implemented, one might be able to form hypotheses regarding human consciousness or be able to discover discrepancies that could be remedied in future versions of the system.

The particular theory of consciousness that has been partially modeled in the neural schema mechanism is Bernard Baars' global workspace model (1988, 1997). This theory puts forward the claim that human cognition is implemented by a multitude of relatively small, generally unconscious, special purpose processes. Within this multi-agent system, groups of such processes, called coalitions, compete for entry into a global workspace. The workspace serves to broadcast the coalition's message to all the unconscious processors, in hopes of recruiting other processors to help solve the current

[7] A more complete implementation of this theory of consciousness is described by Bogner (1999).

impasse or to take note of the current novel situation. The number of processors that can be in the workspace at any one time is limited but not set at a specific number.

Even though in humans consciousness plays an important role in solving problematic situations, it is easy to recognize from one's own experience that consciousness is not only used in these circumstances. To the contrary, humans are usually conscious at any given moment without, necessarily, being in the midst of solving some unresolved issue. This is where attention comes into play. In the theatre metaphor presented by Baars, the spotlight of attention is the method by which coalitions get into consciousness. However, the theory does not dictate what criteria should go into deciding which coalitions are attended to, nor does it dictate that all coalitions that get the spotlight of attention shined on them must be a novel situation or problem, only that novelties have an increased probability of entry.

Attention in the neural schema mechanism has a similar purpose to the one created for the original schema mechanism by Foner and Maes (1994); namely, to reduce the computational requirements of the system. Ours, however, works by using a combination of the change in a node's activation over time, the magnitude of desirability flowing through the node, and the node's association to other nodes recently in "consciousness." Each node in the network has a degree of "consciousness." In this way it can be thought of as similar to activation. All node types, with the exception of action nodes, are candidates for inclusion into "consciousness." The first two elements that make up the calculation for the degree of "consciousness" in any given node are determined without respect to the node's relationship to any other node. First, the slope is calculated for every node in the network. Slope is defined here as the absolute value of

the average change in activation over the last five cycles, plus the absolute value of the desirability at the node multiplied by a desirability factor. The desirability factor is used to balance attention between highly active nodes and nodes deemed important in obtaining or avoiding goals. The resulting slope value is proportional to the rate of activation change in the previous five cycles modified by the desirability level. Since some pre-activation of expected item nodes will occur (priming), the nodes with the highest change in activation values will, most likely, be those that are least expected. Therefore, this portion of the equation gives a measure of the node's novelty (rate of change of activation) and usefulness in accomplishing current goals (desirability). This value is combined with the amount of "consciousness" that was sent to the node through its links. The passing of consciousness is meant to provide the agent with some continuity of thought from one cycle to the next. Under most circumstances this give the agent something akin to a train-of-thought event though highly novel or unexpected occurrences could force the agent to jump to a new awareness unrelated to the previous contents of "consciousness." At this point the "consciousness" values for all of the nodes in the system are normalized in order to maintain a constant value for the total amount of "consciousness" in the system.

The size of attention (the number of nodes in the spotlight) is dynamic and can change based on the overall state of the system. After the "consciousness" is calculated for every node, the values are totaled and a percentage of that total is noted. The percentage of the total slope is a constant set up at the beginning of a run that dictates how inclusive the spotlight of attention is. A value of thirty percent, for example, would say that only those nodes whose degree of "consciousness" makes up the top thirty

percent of the total would be in the spotlight. The percentage quota system allows the number of elements that get into the spotlight of attention to grow when the system is in a relatively stable state and narrow when the system is experiencing drastic change or strongly pursuing a particular goal. As additional nodes are added to the system, the average number of nodes that are "conscious" in each cycle will tend to increase, although that increase is not necessarily proportional to the total number of nodes in the system. A stable state for attention occurs when activations are not changing rapidly and no one goal (or small number of goals) is being robustly pursued. For example, if the total "consciousness" of the system were spread somewhat evenly over twenty different nodes, then a thirty percent spotlight value might include five or six different nodes. If, on the other hand, just five nodes provided fifty percent of the "consciousness" in the system, then a 30% spotlight value would only encompass two or three nodes. As can be seen from the example, changing the spotlight percentage value effects the general focus of attention while still allowing it to adjust to changing situations. Experimentation needs to be conducted to determine an optimal value for this percentage that maximizes learning by creating enough links between nodes while minimizing the branching factor of the resulting network.

The branching factor of the network is largely determined by the creation of links between attended nodes. Once a set of nodes is selected for "consciousness" by the attention mechanism, two things happen. The first task is to update the links between all of the nodes in the spotlight. A node cannot have a link to itself, and action and item type nodes cannot link to other action and item type nodes. If two nodes, say Node-A and Node-B, had no previous connections between them, then two new links are created, one

with Node-A as input and Node-B as output, and one connecting the two in the opposite direction. This is in contrast to the fully connected nature of the original mechanism. Previous research has shown that an attention mechanism that operates in a similar manner with regards to this first task, reduces the branching factor of the mechanism without significant lose of learning capabilities (Foner & Maes, 1994).

The second thing that happens to "conscious" nodes is that a broadcast is sent out to all schema nodes in the network. The contents of the broadcast consists of a set of item nodes. If an item node is in the spotlight, then it is, itself, one of the item nodes broadcast. If, on the other hand, attention is broadcasting information for a schema node or goal node, then that node's context set is broadcast. For goal nodes, we would say that their goal set is broadcast. In total, the broadcast will contain all of the item nodes represented by the coalition of nodes in the spotlight. Schema nodes respond to the broadcast if any of the items transmitted correspond to any or all of its result set. The schema node responding to a "conscious" broadcast sets an internal flag that denotes that it is responding and increases its activation proportional to the percentage of its result set represented in the broadcast. Responding to a broadcast does not automatically guarantee selection for activation, but it does increase the likelihood. The flag denoting a node's response to the broadcast is an overriding factor to the applicability rule for action selection. It is in this way that an agent can discover new paths to a solution or goal state that did not previously exist or were not active highly enough to become chosen for activation. Selecting schema nodes whose result sets match the desired goal under these special circumstances even when the context is different or unknown allows the agent the possibility of quickly learning the correct context or alternate context for achieving the

goal. For example, if the schema node chosen for activation by a response to the broadcast has no context at that moment then its selection serves to decrease the time necessary to discover the correct context due to its increased likelihood of activation. On the other hand, if the chosen schema node does have a context and the action turns out to be successful in bringing about the expected result, then the mechanism has the possibility of creating, in very few steps, a new schema node with a complex context solution to arriving at the broadcast goal. Otherwise the system would have had to build up the solution from a series of schema node creations.

The general formula for attention serves a number of different purposes. First off, it increases the likelihood that unexpected occurrences will become "conscious". Because of the way activation is spread within the system, the mechanism displays the psychological phenomenon of priming and, along with the "conscious" broadcast, the phenomenon of habituation. Priming is a well-studied effect in the human brain where the future activations of concepts can be influenced by similar currently active concepts. The classic example of this effect is the disambiguation of a word with multiple senses. In the stereotypical version of these experiments, a subject is shown the word "stream" and is then shown the word "bank" and is asked to provide the meaning for "bank." In this situation, the subject is likely to give the sense of the word "bank" relating to "river bank". Another subject first shown the word "money," however, is likely to respond with the sense of "bank" relating to a financial institution. The notion is that concepts related to the first word presented receive some activation, which influences future activations toward those related concepts. It is likely that unconscious expectations are manifestations of this same process. When a person performs an action that they have

performed numerous times in the past, such as turning on a light in their house, activation flows to sensory nodes that should feel the light switch at the next moment. When that state of the environment obtains and those senses are triggered, the activation of the sensory nodes rises only slightly; the expectation has been confirmed and the action completes unconsciously. It is only when something unforeseen occurs that a sensory node must jump from a relatively low activation to a high activation (or vise versa). This unexpected change in activation is the first part of what the attention mechanism uses to decide what becomes conscious.

Action automation occurs as a necessary precursor to reliable priming. Before the links between two nodes can become strong enough to prime a behavior unconsciously, the links will have had to have been used a number of times, most likely, consciously. Repeated successful use of a given chain of schema nodes increases the weight of the links between them, causing a chain, that initially drew the spotlight of attention, to prime itself in such a way as to minimize the change in activation between the selections of schema nodes in the chain. At the point where all of the schema nodes in the chain can be activated in sequence without drawing the spotlight of attention, the chain has been automated. While it is probable that schema nodes, which activate in sequence but are weakly connected (through their result/context item nodes) will be "conscious" due to their changing activation slope, it is not assured. Enforcement of this policy would better fit Baars' psychological theory (1997) but is not currently implemented.

The other element in the formula for attention is the desirability of a node. As shown previously, desirability is the measure of how much a given node is in service of the agent's currently desired goal context (or contexts). For this reason, the addition of

desirability to the formula increases the degree to which the agent's current goals are represented in "consciousness."

The use of "consciousness" does not always change the behavior of the system. To the contrary, for most situations the mechanism would have chosen the same set of schema nodes in the same order due to the other factors already laid out. At times such as these, one could say that the agent is "consciously" aware of what it is doing or experiencing, but that "consciousness" plays no discernable role in the agent's behavior. However, under circumstances where a path is not clear or where two (or more) separate chains of schema nodes must converge on a goal state, "consciousness" often provides a solution.

9 Results

Neural schema mechanism was tested primarily in the Wumpus World environment although the system was also run in other environments during debugging and initial testing stages. The most extensive tests, however, were conducted within the Wumpus World and they will be the focus here. The elements that these tests are meant to elucidate involve:

1. The ability of a neural schema agent to learn sensory-motor correlations within its environment
2. The ability of a neural schema agent to act appropriately within its environment based on its primitive goals
3. The ability of a neural schema agent to learn intermediate goals and make use of them
4. Whether there is any discernable benefit to the "conscious" broadcast

To do this, several quantitative pieces of data were tracked during each run of the system. These included the gain value (discussed below), the number of nodes in the system, the average number of links for each node, the CPU usage, and a complete listing of each node that was learned and when it was learned. Although all of these measures tell us a bit about how the mechanism is performing, the two most important gauges are the gain and the log noting what the agent has learned.

9.1 Testing methods

First, let's discuss how one can discern the mechanism's performance with regard to the four issues listed above. In deciding how well the agent has learned sensory-motor correlations for its environment, the best way to do this is to carefully examine the log

which gives a complete description of what the agent learned and when. The typical entry in this log looks like Figure 9.1. Adding a version number on the end of the name of the node from which this new one has been spun off, its parent, generates the name of each new node. The version number denotes how many spin-offs from the parent node have been performed before this spin off. For example, "`schemaNodeClimb.3`" says that this is the fourth spin off from the "`schemaNodeClimb`" schema node. This naming convention allows for the genealogy of most nodes to be discernable using only its name. The log also lists all of the elements that define a schema node including its action, context set, and result set along with a tag, "`** NEW **`," that denotes which piece of this schema node is a result of this spin off. In addition to schema nodes, the log also lists new item nodes and any nodes that have been deleted through the garbage collection processes. By examining this log, one can distinguish how complete the agent's knowledge is on any given cycle and whether the agent knows a path to a particular goal state. It is also possible to describe the specific knowledge construction process that the agent went through.

The next measure involves determining how well the agent is performing at each time step. This is not as straightforward as it might seem. For a neural schema agent a

```
New Node created: schemaNodeMoveNorth.1.0 of
type SCHEMA on cycle 2422
    action:
        moveNorthActionNode
    context:
        sensePos20 = on  ** NEW **
    result:
        senseBreezeNode = on
```

Figure 9.1: A typical entry in the new node log

gain value is calculated that ranges from 0 to 1. This gain value needs to represent both how well the agent is doing at achieving the goals it desires and how well the agent is doing at avoiding the goals that represent negative states. To arrive at this value, first, the total possible positive desirability values in the primitive goal nodes are calculated along with the current level of positive desirability values actually present, and an amount for the current positive values is converted into a percentage of what is possible. The same process is undergone for the negative values. The resulting percentage of the negative value is subtracted from the positive value and this result is normalized to a value in [0, 1] (see Equation 9.1). Most of the time, the agent is neither achieving a positive goal nor in a state represented by a negative (or avoidance) goal. This means that the typical gain value is somewhere around 0.5. If the agent is in a negatively valued state, then the gain will go down. However, if the agent is in a positively valued state, then the gain will go up. This rather complex calculation for gain is necessary since the agent could be in a state that is recognized to varying degrees as being of both a positive and negative value.

Even though the gain provides an accurate picture of the agent's situation at any one moment, a graph of the gain values over time could be misleading depending on the

$$\frac{1}{1+e^{-5x}}$$

Where x is:

$$averagePositive - averageNegative$$

Equation 9.1: Gain calculation.

agent and environment in question. Such a situation would be one where the agent must endure a somewhat negatively valued state in order to reach a more highly valued positive state. Viewing the graph of the gain values as a performance measure for an individual run in this situation would, therefore, show a consistent drop in performance before the increase in performance. This explanation is meant to note that the gain calculation, while being the closest internal measure for the performance of the agent, is merely an approximation and should be considered along with the agent and its environment to determine how accurate it is. For this reason, the graphs of gain values that will be shown and used as examples later in this section were generated over multiple runs within a Wumpus World configuration that did not require the agent to assume any negative states in order to reach the primary goal state. Averaging over multiple runs tends to dampen anomalies within individual runs in favor of prevailing consistent trends.

Another point that should be noted regarding the gain value is that it only takes into account the primitive goal nodes and does not include any intermediate goals that the agent might have created during the run. The reason for this is that the desirability values contained in the agent's intermediate goals are constantly in flux and may not be accurate at any given moment. To include these non-primitive goal nodes would give us a measure of how well the agent *thinks* it's doing (also an interesting measure), but does not provide a good objective appraisal.

The third item that was tested for involves whether the system learns intermediate goals and appropriate values for those goals. This value is denoted by the "delegated value" in the learned goal nodes. Unfortunately, just graphing these values or some

function of these values over time would not yield any useful information since there is nothing to which they can be compared. In addition, the agent may maintain incorrect values for some period of time depending on its experiences. This fact is completely expected if you compare this with humans; it is not risking a great deal to conclude that no human has a perfectly balanced and correct set of intermediate goal values. Instead, humans have values that, for the most part, allow us to behave appropriately at least to the degree that can be judged by the other individuals around us. Essentially this same method is used to judge the learned goals of a neural schema agent; an external observer watches the agent's behavior to determine if non-primitive goals are generally appropriate and that they are being followed. Of course, the external observer in this instance also has the ability to actually see, explicitly, what the learned goals are and the delegated value that has been imparted to them, but the effect that these goals have must be discerned by noting the agent's actions under situations where the learned goal or goals apply.

The final element tested for involves whether the use of a "conscious" broadcast provides any discernable benefit over just the use of an attentional mechanism. To quantify this element, several runs were made in identical environments using identical initial network configurations. Some of the runs were performed using the full neural schema mechanism including the "conscious" broadcast while some of the runs had the broadcast turned off. More specifically, the broadcast was hard-coded to only send the empty set, meaning that no node would respond. To compare the performance of the agents in the two different types of runs, gain values were graphed and compared.

9.2 Evaluation

During the course of several runs of the neural schema mechanism within the Wumpus World environment, over 30,000 lines of text was generated just in the log files that contain the learning information of the system (see Figure 9.1 for an example entry). The examination of these log files reveals that the mechanism is indeed learning schema node constructs that are correct for the given environment. Unlike the original schema mechanism, however, the order of learning in a neural schema agent is less predictable. The main reason for this stems from the fact that the actions and senses of the agent

```
New Node created: schemaNodeMoveWest.0.0 of
type SCHEMA on cycle 1211
    action:
        moveWestActionNode
    context:
        sensePos31 = on  ** NEW **
    result:
        sensePos30 = on

New Node created: schemaNodeMoveWest.3 of type
SCHEMA on cycle 1337
    action:
        moveWestActionNode
    context:
    result:
        senseBumpWestNode = on  ** NEW **

New Node created: schemaNodeMoveNorth.2 of type
SCHEMA on cycle 1263
    action:
        moveNorthActionNode
    context:
    result:
        sensePos30 = off  ** NEW **
```

Figure 9.2: Some examples of schema nodes created early in runs. " NEW **" denotes newly added elements.**

```
New Node created: schemaNodeMoveNorth.2.2.0 of
type SCHEMA on cycle 54799
    action:
        moveNorthActionNode
    context:
        sensePos12 = on
    result:
        senseStenchNode = on
        sensePos02 = on  ** NEW **

New Node created: SchemaNode86.0.1 of type SCHEMA
on cycle 98783
    action:
        ActionSchemaNode86
    context:
        sensePos11 = on  ** NEW **
    result:
        schemaNodeMoveNorth.8I.3 = on
```

Figure 9.3: Examples of more complex schema nodes. " NEW **" denotes newly added elements.**

directly shapes what is attended to via the attention system. Since the actions of the agent at the beginning of a run is, essentially, random and learning cannot occur without links, the order of the agent's knowledge acquisition is based, primarily, on the probability of situations occurring randomly.

Obviously some situations are much more likely than others so that it is probable that the agent would learn schema nodes relating to these environmental states early in a run. For the Wumpus World environment such schema nodes turned out, not surprisingly, to be those that related knowledge about the initial position of the hunter agent. Some examples are given in Figure 9.2. As would be expected, the schema nodes created tend to represent increasingly more complex states while maintaining the validity of those states.

As can be seen in the examples given in Figure 9.3, the mechanism begins to learn about co-occurring results and about the results of learned actions. The generic names for certain nodes such as `ActionSchemaNode86` denotes a node that was created in response to the spin-off of a schema node with a novel result set. In addition, it learns about how to bring about abstract states represented by synthetic items. The "I" in the name of an item node inside of the version number area denotes that this is a learned item whose original host is the beginning part of the item node's name.

All of this information is useless, however, if the agent can't use it to bring about desired states. To determine if the agent is accomplishing anything useful with its knowledge, the gain value, as described previously, is graphed over multiple runs with the same initial conditions. This graph for the Wumpus World environment is shown in

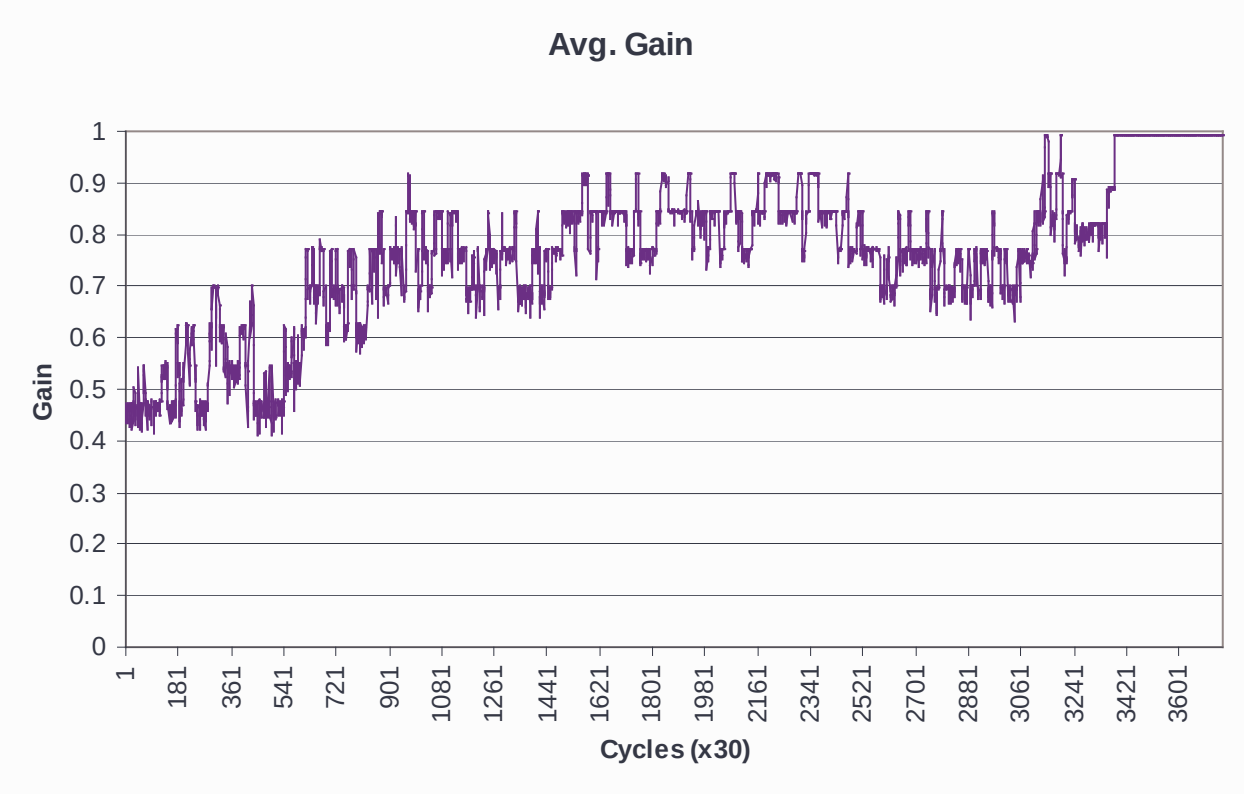

Figure 9.4: Average gain produced over multiple runs

Figure 9.4. During each run the gain is tracked for thirty cycles, averaged, and that value written to the log file. For this reason, the labels on the x-axis are the actual cycle number divided by thirty. The "jagged" nature of the graph is due to the way that the gain values fluctuate by a relatively large degree in any given run. What is more important to note, however, is the trend towards higher achievement. The nearly flat line towards the end of the graph is evidence to the fact that all of the runs had achieved near perfect performance at this point. Perhaps the most interesting development visible in this graph is the apparent reduction in performance at approximately tick-mark 2521 and then the subsequent increase in performance to levels even higher than previous at approximately tick-mark 3100.

A clearer view of how this plays out in a single run is shown in Figure 9.5 where gain values above 9.5 are considered to have achieved the goal. Line C in this graph, representing 56 percent of the runs, shows a clear drop in the number of goal achievements for a period of time with a subsequent return to high performance later on. An explanation of why this might have occurred doesn't become clear until this graph is

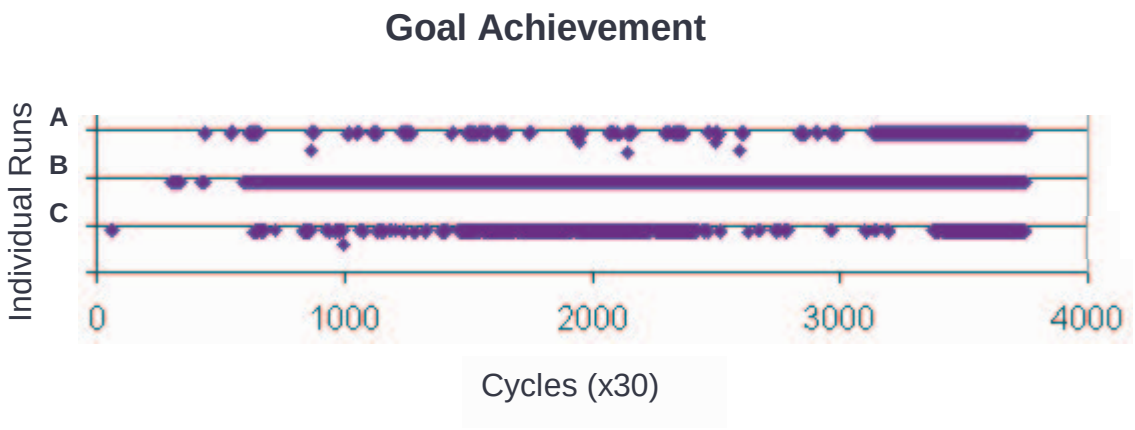

Figure 9.5: Rates of Goal Achievement for typical runs. Line A represents the type of goal achievement seen in 32% of the runs, B represents 12% and C represents 56%.

compared to other information available. For one, a graph of the average total nodes in the system shows some additional data that suggests that the mechanism (in the Wumpus World) goes through one major and one minor jump in the total number of nodes (Figure 9.6).

Keep in mind that the acquisition of knowledge does not immediately translate into better performance since the agent may have to decide over time whether to use the new knowledge or not based on their reliability. So, the first large jump in total nodes, signaling a kind of mini-renaissance of learning, actually results in the mechanism performing more poorly for a period of time. This situation is not reversed until the onset of the second, more limited, jump in new nodes. In looking at the log describing the new nodes learned in these two periods, it became clear that the first major jump resulted in

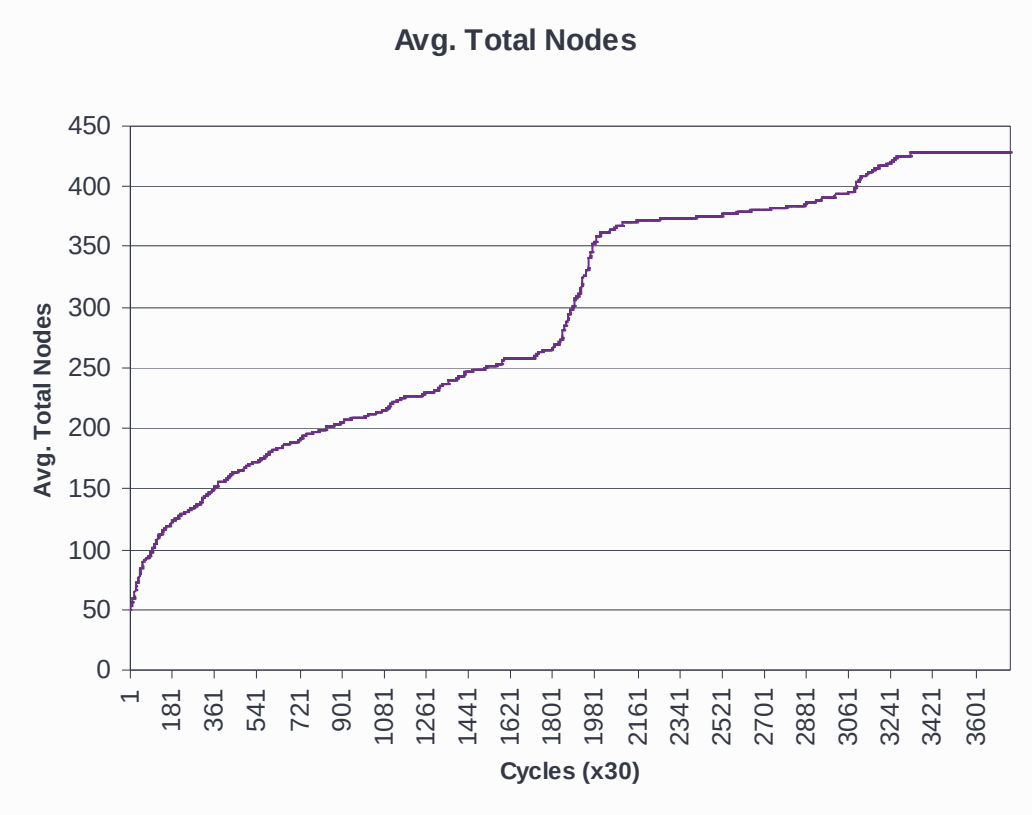

Figure 9.6: Average total nodes over several runs

schema nodes for general situations that worked for many situations being refined into more specific and more reliable schema nodes. The problem was that while the more general schema nodes would not necessarily be chosen in the situation in which they applied due to lower reliability, the more specific schema nodes would almost always be chosen. The drop in performance occurred because these more specific nodes were not yet specific enough and tended to have disastrous results if misapplied. It wasn't until the second jump in learning occurred that these new schema nodes were refined even further to allow the agent to perform correctly in nearly all situations.

It is important to note, as shown in lines A and B in Figure 9.5, that this drop and subsequent increase in performance did not manifest itself in all runs; it actually occurred in a little over fifty percent of the runs. The other runs had two major tendencies. Either the mechanism would learn general rules that worked the majority of the time and quickly get to a near perfect performance, or the mechanism would never have the initial increase in performance but would eventually arrive at almost the same set of schema nodes as the agents did in the runs that did have the initial performance increase. Even though the analogy is not quite perfect, this phenomenon is akin to the way that children often learn to use irregular verbs in the English language. It has been shown that children will generally learn the correct usage of irregular verbs only to then begin to use them incorrectly once they learn the rules for conjugating verbs. In these instances the children will attempt to conjugate irregular verbs until they learn that these verbs are exceptions to the conjugation rules (Pinker, 1995). Just as with the neural schema mechanism, there are three stages in this process. The first involves what might be considered a general rule of "verbs go in this spot in a sentence" with the child having a memorized list of

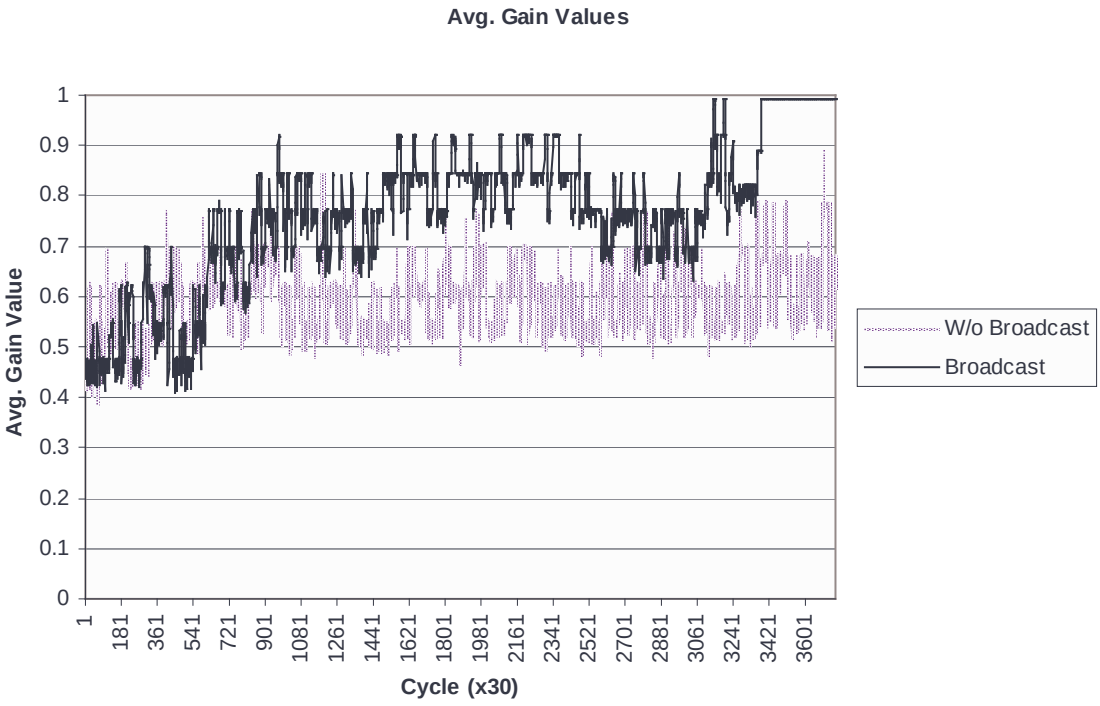

Figure 9.7: Comparison of gain values with and without a "conscious" broadcast

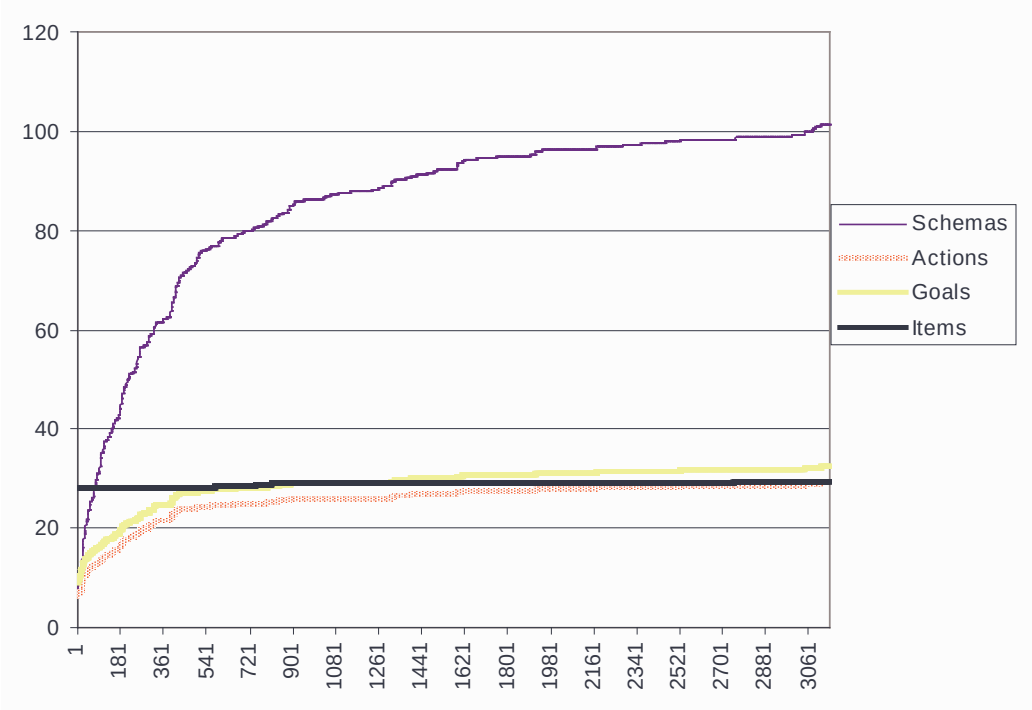

Figure 9.8: Total nodes created divided by type

verbs that it knows. The second stage is where the child learns more specific knowledge about the use of rules to conjugate verbs and uses the rules in some inappropriate situations. The third stage is a further step in specificity learning for the child where they begin to only apply the conjugation rules under certain circumstances.

Another interesting point with regard to this phenomenon leads directly into the question of whether there is a discernable benefit to a "conscious" broadcast. For neural schema mechanism, "consciousness" consists of two components that can be partially separated, an attention mechanism and a system-wide broadcast. Although the system can run without the broadcast, the attention mechanism alone would not constitute "consciousness." However, many systems over the years have had attentional components. The tests described here are meant to explore whether a fully "conscious" system (one with attention and a broadcast) is appreciably different than one with only an attentional subsystem. First off, in the mirror runs performed without the "conscious" broadcast, the learning dynamics described in the previous paragraph never occurred. It is not clear that this particular factor can be deemed a benefit even if it does mimic some element of human psychology. From a performance standpoint, however, there is a definite advantage to having a "conscious" broadcast as can be seen in Figure 9.7. In addition, the mechanism does not reach the same level of either total learned nodes in the system or links per node as shown in Figures 9.9 and 9.10 respectively. It seems clear from these results that the "conscious" broadcast increased the rate of learning and that the information learned was useful in bringing about the agent's primary goals. In particular, the "Links per Node" graph in Figure 9.10 points to the fact that the mechanism, using "consciousness," was able to make connections to knowledge

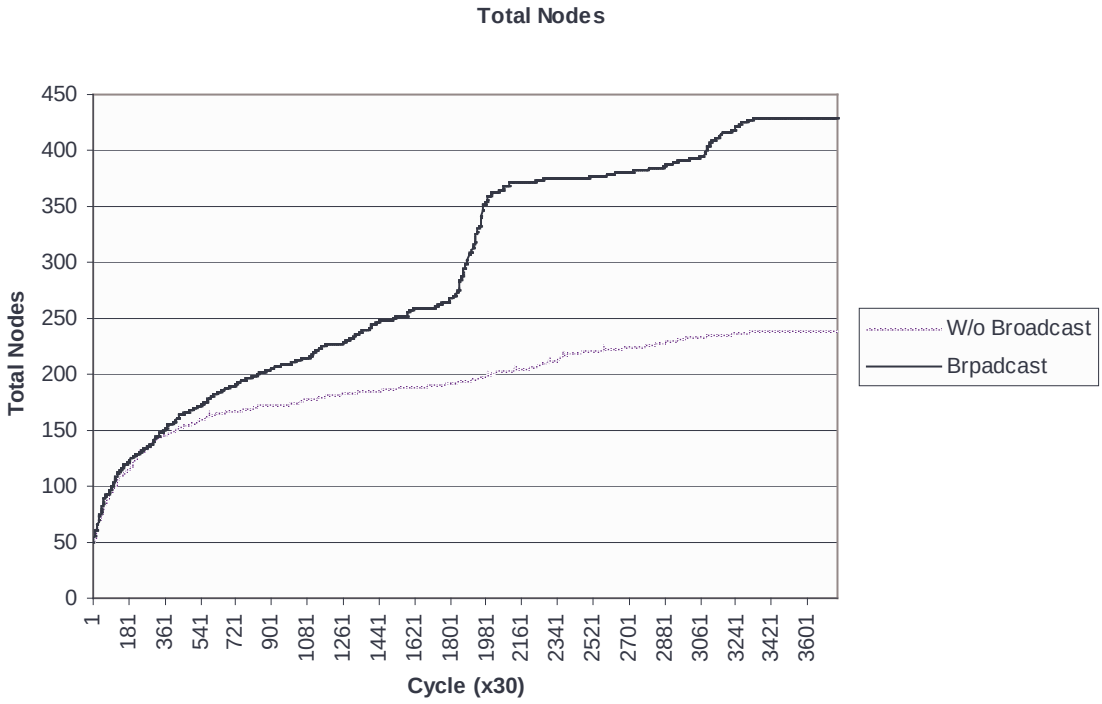

Figure 9.9: Comparison of total nodes with and without a "conscious" broadcast

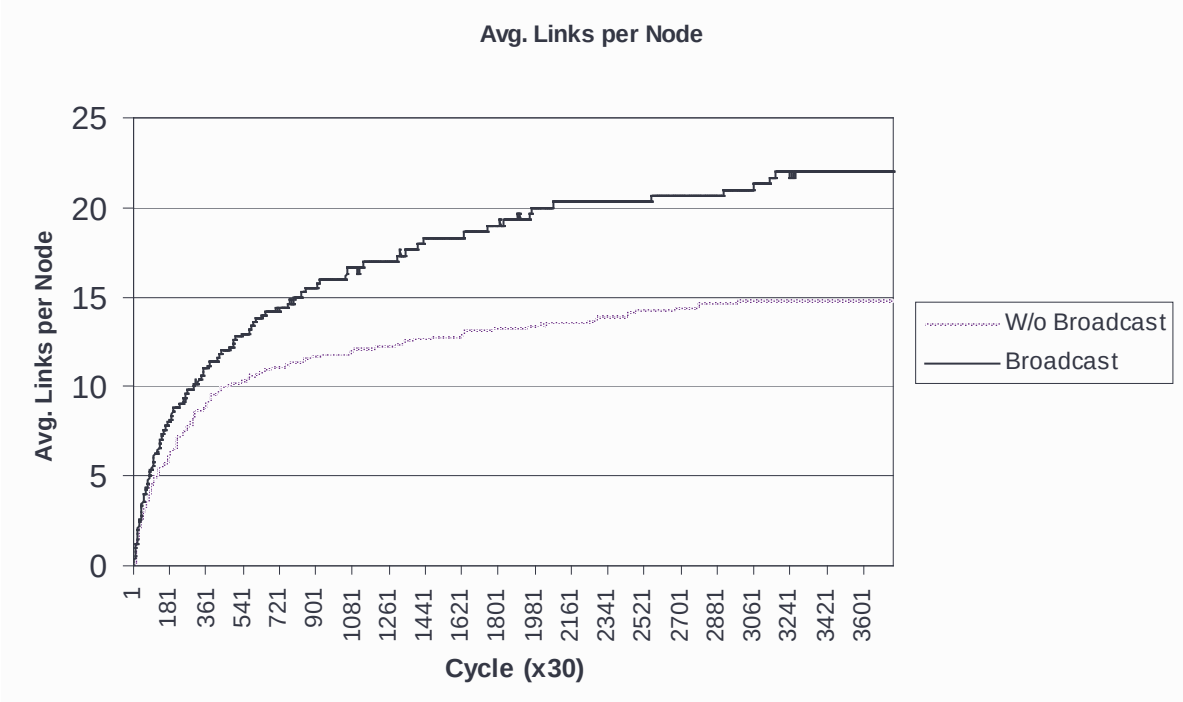

Figure 9.10: Comparison of average links per node with and without a "conscious" broadcast

structures that would not have been correlated using only an attention mechanism.

An additional speculation could be made that the learning that is taking place in the Wumpus World environment and displayed in the previous graphs is due to a significant increase in abstract constructs such as learned item nodes or learned action nodes. Figure 9.8 breaks down the total number of nodes created during several runs based on their type. From this graph and examination of the node learning log file it is clear that the increase in the performance of the system is mostly due to an increase in the schema nodes, although a small number of key learned item nodes also contributed. Like the original mechanism, the neural schema mechanism is still quite computationally expensive. The current implementation has not been optimized for performance beyond what was dictated by the larger architectural issues discussed previously. It is quite probable that the mechanism could be improved computationally if an analysis were done that would show where the cycle manager software module could be condensed. Appendix B shows the steps taken by the manager module. Some of the loops required in this part of the mechanism could, most likely, be combined with other loops to decrease the computational requirements. It might also be possible to assign separate threads to different loops so that they could be run in parallel.

Even so, the new mechanism has successfully run on a single 700Mhz CPU computer where the original mechanism ran on 16,000+ processors of a Thinking Machines CM2® system. It is unclear how much of this speed-up is due to the much faster computers available and how much is due to the implementation architecture. On a small number of occasions the system has been run for approximately 350,000 cycles. By then it had learned over 700 nodes and averaged about 45 links per node. At this

level, the system continues to function albeit at a pace that is approximately half its default fastest speed. Although actual CPU usage is much smaller initially, the default speed for a single cycle is currently set at 100ms. On each cycle, the system keeps track of exactly how many milliseconds elapse between the start of the cycle and when all of the work has been completed for that cycle. If the actual time used for the tasks in that cycle is less than the currently set default, then the system sleeps for the remaining amount of time. For example, if a given cycle required 30ms to complete, then the system would sleep for the 70ms left in the default 100ms given for each cycle. The actual CPU time used for a series of similar runs is shown in Figure 9.11. The mechanism automatically increases the default cycle time if a cycle requires more time to

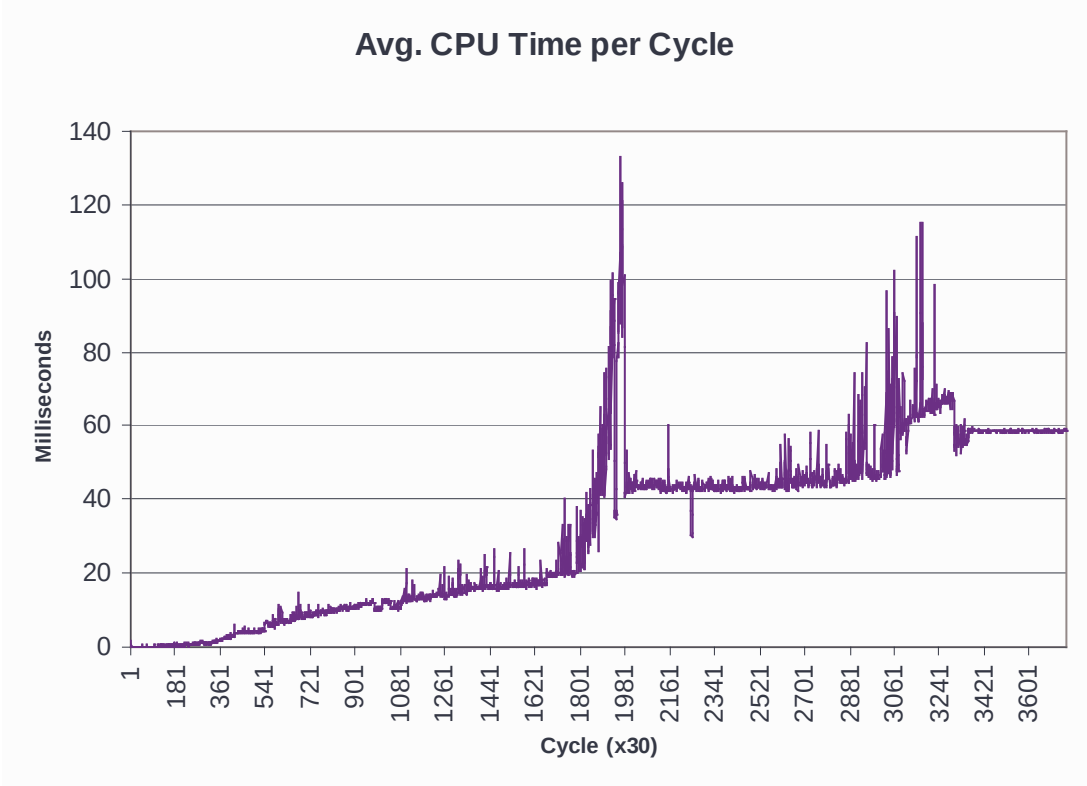

Figure 9.11: CPU time (in milliseconds) used per cycle

complete than the currently set default. At it slowest point, the system had increased the individual cycle time to about 210ms. The increase in computing power expected in the near future may allow a large-scale neural schema mechanism to be deployed and tested in a real-time environment, but currently it is not clear if such would be possible for anything more than small to moderate size environments.

The final element that was tested centered on the mechanism's ability to create intermediate goals and to learn reasonable values for those goals. For reason expressed previously, this is a highly subjective measure requiring a human observer to interpret the behavior of the agent and to examine the dynamic goal structure during the coarse of a run. It might be possible to conduct human trials that could be used for comparison, but has not been done at this point and would, ultimately, still require a human interpretation.

Two questions need to be answered in order to decide whether the goal structure that a neural schema agent creates is reasonable: (1) does the agent avoid or pursue states that are not part of its primary goals in a manner that would be reasonable given what the agent knows and (2) does a particular learned goal tend to acquire appropriate delegated value over time? Watching the behavior of the agent in the Wumpus World environment seems to show that the agent does tend to avoid states that are likely to result in negative states. For example, the agent will eventually begin to avoid states where it senses a breeze or smells a stench. At an earlier point in the run the agent will often enter those states only to quickly return to the previous state. Later on, it will simply not go into those grid positions except on rare occasions. Close examination of the learned goal nodes during a run show clearly that these structures are acquiring reasonable values. In other words, a goal node that represents a higher likelihood of a negative state acquires a

negative delegated value. Goal nodes that represent a higher likelihood of a positive state also acquire appropriate positive delegate values and tend to manifest much quicker than those goal nodes for negative states. The reason for this is that the mechanism considers a state only as bad as the value for the best state that can be reached within a single action. This means that a state such as one where the agent smells a stench does not tend to acquire the value of the worst state from it which would be the agent being eaten by the wumpus, but the best state of reversing direction back to some neutral state or even, possibly, to proceeding to a state that is also safe and happens to be within a step of the gold. Of course all of this is based on probabilities and the experience of the agent. An agent, for instance, that randomly chooses the wrong action on several occasions when in a state of sensing the breeze will still acquire a strongly negative delegated value for this state even if there is the possibility of an action that would render good results. As far as the agent is concerned, it only received a good result, say, once out of eight times. The agent will maintain a negative value for this state until it can learn how to achieve the good result from this state and has successfully performed that action on several occasions.

A side benefit of the current implementation lies in its interface. For ease of use purposes, the system was designed to allow for the reading and writing of network definitions via an XML document. This means that the initial network for a given task can be provided in XML format from an outside system such as a graphical design tool and, more importantly, the system can write the complete state of a network at any time to an XML document. This resulting document could be used in an analysis tool or even as the specification to a static hardware or software version of the network. Such static

versions could be optimized for speed and compactness. The only caveat about such a use of the mechanism is that the simulation environment that it is provided with must be as similar as possible to the actual environment into which the system will ultimately be placed.

Finally, while the neural schema mechanism was demonstrated here as a stand-alone system, this need not always be the case. In fact, the mechanism may be more useful as part of a larger system in which sophisticated sensors are used as the inputs to a neural schema network. For example, more conventional neural networks could be used to recognize elements of a visual image provided by a camera on a mobile robot. The neural network's output could then be used as sensory input to the neural schema network as a continuous measure of the existence (or non-existence) of a set of elements in the image and their precise locations.

10 Future Directions

Neural schema mechanism is by no means a complete model of mind. There are several facilities of human minds for which this system does not account. However, it may provide the foundation upon which such capabilities could be incorporated. The areas that neural schema mechanism could be extended into include memory, metacognition, imagination, and probably others that have not yet been considered. This section describes how these extensions might be accomplished and what the implications could be.

10.1 Memory

One component in neural schema mechanism whose absence seems glaringly obvious is memory. The system has no way, aside from what is represented as procedural knowledge in the schema nodes, of remembering a past situation or any associations that could be of use. This limits a neural schema agent in the kinds of environments and tasks that it could be expected to adequately deal with. In particular, environments that require deictic representations of sensory input would be difficult for the agent to learn. This statement is dependent, of course, on how distinctive the various world states are relative to the agent's sensory apparatus. For example, if we were to remove the sensory items in the Wumpus World environment that gave names to the grid positions, then many of the world states would look exactly the same from the agent's perspective. Under these circumstances the agent would need to learn a sequence of actions connected via undifferentiated states. In its current form, the mechanism has no way of learning sequences except through chains of schema nodes connected through common result-

context match ups. The lack of differentiating world states would preclude the learning in this type of situation.

Adding memory, however, would mitigate this problem. This could be done by simply adding additional item nodes whose activation and state represent the previous states of the particular primitive item nodes that register the current world state. Any number of additional levels of this type of memory could be added to represent however far back in the past one might want to represent. An alternative, and generally more reasonable possibility, would be to only update the item nodes of memory that correspond to item nodes whose states have changed in the past cycle. This would allow the normal decay of activation in the memory nodes to "forget" over time. Another modification of this alternative would only change those memory nodes corresponding to item nodes that were brought to "consciousness" in the last cycle.

This type of memory could not be considered long-term memory but more of a working memory. The agent, using this working memory, could then make connections with the memory item nodes in order to learn correlations not only with immediately occurring results or contexts, but also with states that occurred in the relatively recent past.

A long-term memory could be added using some additional mechanism such as Sparse Distributed Memory (Kenerva, 1988) or a neural network designed for memory retrieval such as a Hopfield network (Hopfield, 1982). Using one of these mechanisms would entail feeding the states or activations of the primitive item nodes into them and then having a separate set of primitive item nodes whose activation was reflective of the

output of these systems. These memory output item nodes could then be used in the context and result sets of schema nodes.

10.2 Metacognition

Metacognition is the ability for an entity to think about and modify its own thinking. A somewhat limited version of metacognition could be implemented in neural schema mechanism by using a second neural schema network whose item nodes registered the states of schema nodes and, possibly, other item nodes in the first neural schema network. This format is akin to the B-brain notion of metacognition proposed by Marvin Minsky in which an A-brain, which interacts directly with the environment, is watched and influenced by a B-brain (1986). In the neural schema version of the B-brain, in addition to allowing for all of the normal functions of the mechanism, primitive item nodes would have to be created in the B-brain during the running of the agent whenever a new schema node was learned in the A-brain.

It is also unclear exactly what actions such a B-brain would perform. There could be a set of global variables that effect how the A-brain works which could be altered by actions of the B-brain. There are a good number of potential variables that are, currently, set up as constants in neural schema mechanism that could become accessible to a metacognitive B-brain. Also, a degree of trash collection within the A-brain could be carried out by the B-brain in order to maximize resource utilization.

These suggestions are only the most obvious activities that could be performed by metacognition in a neural schema agent and are far from complete. For instance, would it be possible for the B-brain to look not only at the A-brain functions but also at its own

functions and make adjustments there? It is not clear if neural schema mechanism would be able to handle this type of high-level thought, but it would be worth exploring.

10.3 Deliberation

Neural schema mechanism does not explicitly plan actions in advance. Instead it creates chains of knowledge structures that predict and explain the results of actions under given contexts. This could pose a problem to an agent in situations where two moderately reliable paths have been discovered to some goal state but one path is narrow and risky, meaning that there is no margin for error and if an error is made then the result is dire, and the other is longer but safer. In instances like this, neural schema mechanism would likely attempt the more risky path since it is contains fewer steps to the goal state. If, however, the mechanism could somehow "virtually" take both of the paths without actually having to travel down them, then the agent might make a different choice. In particular the agent would need to note the results of minor deviations from the shorter path as apposed to the longer, safer path.

Deliberation could be accomplished in neural schema mechanism by providing a "virtual" mode. This would be a mode where feux-activation and feux-desirability would be sent along the links to schema nodes and they would react as though it were real activation and desirability. All of the primitive actions would not actually trigger their actuators and non-primitive actions would trigger feux-desirability to be sent from their respective goal nodes. The results of a schema node's execution would be the feux-state change of the item nodes in its result set, thereby simulating the expected result of the action's execution. Note that the results of other schema nodes with that same action would also need to be resolved in this virtual manor. Finally, the results of a schema

node's virtual execution would hold only with the probability dictated by its reliability. In other words, if a schema node is chosen for virtual execution, then a random number is generated to determine if the schema node's result will occur exactly as stated. If the random number is below the reliability of the node, then the results virtually obtain without alteration. However, if the random number is above the reliability, then one of the schema node's result item nodes, also chosen at random, will not change its virtual state. This virtual exploration of paths might need to occur several times in order to make a reasonable guess at the likely outcome.

Another important use of this type of deliberation would be to allow the system to explore possible paths that have not been traveled enough for a reliable chain to have been created or even to think about a path that has never been considered previously. If we were to add one additional function to the previous description, then the mechanism would be able to "imagine" in this way. For those item nodes whose states are not explicitly predicted by the current virtual activation of the system, we need to determine a possible activation. This could be done by allowing these item nodes to add up the relevance values for all generic links coming to them for which they are potential results and noting the sign of the total. If the sign is negative then their virtual state should be set to *off*, otherwise it should be set to *on*. Some of the item nodes will not change their state based on this calculation since they are already in that state; others, however, will change their virtual state and could provide insight into a previously unknown possibility.

While it might be possible to allow the mechanism to learn while in this virtual mode, it might be difficult to reconcile the "imagined" probabilities with the experienced

ones. Even so, this type of deliberation could prove quite useful and should be examined more closely.

11 Conclusion

Neural schema mechanism is an attempt at creating an autonomous agent control structure such that an agent using it would perform well even when placed in an environment for which it has little or no prior knowledge. From an initial state consisting only of a set of actions, sensory inputs, and primitive goals/drives a neural schema agent can discover what its actions do and how to achieve its goals. It learns about its environment through its own experience and makes value judgments based on its own internal drives. There are still several avenues of experimentation and extension that would be of benefit.

There are two drawbacks to the neural schema mechanism in its current state. First, the learning rate of the system with regard to the rate of new knowledge created is slower than had been hoped and may well be slower than other methods although direct comparison is difficult and has not been attempted as of yet. The second drawback is the computational complexity of the system. The network within a run does not grow exponentially fast due to the utilization of several methods that suppress redundant or obviously fruitless paths and due to the use of trash collecting techniques that remove unnecessary nodes after they have been deemed erroneous or very rarely used. Even so, the system does display a small but noticeable slowdown on a PC with a 933Mhz processor and 512MB of RAM when the network begins to get large (around 700 nodes). The mechanism has not yet been run to its breaking point but the question of scalability is a serious issue. If a relatively small environment such as the Wumpus World can tax the system computationally, then it is unclear if larger environments could be handled in real-

time. There is always the possibility that new hardware will render this issue moot, but optimizations should, nonetheless, be explored.

Despite these issues neural schema mechanism has shown promise from both an artificial intelligence and cognitive science perspective. In essence, a neural schema agent, like a human child, creates its own reality. It takes agent autonomy one step further, allowing an agent to not only decide on the path to its goal, but also to create new goals. In addition to being inspired by human psychology, neural schemas demonstrate clear advantages to the use of those cognitive phenomena. The way in which consciousness, in particular, has been modeled allows the mechanism to display behavioral characteristics similar to what the psychological theory describes as benefits of consciousness (Baars, 1988, 1997). The mechanism has shown the ability to choose appropriate or reasonable actions more often than randomness would dictate in situations where complete knowledge of the world in that state had not yet been reached. In addition, while exact statistics have not yet been gathered with regard to the learning rate increase expected with "consciousness," the difference in performance is quantifiable.

The neural schema mechanism's motivational system has shown that it has the ability to not only pursue multiple goals, but to do so in such a way as to reduce its own risk. For example, in the Wumpus World environment the hunter agent will often avoid squares that trigger the "stench" perception (those squares adjacent to a wumpus) even though the "stench" percept was not one that had been attached to an innate avoidance goal. No external entity had to tell the hunter agent that a "stench" represented something bad for it; the agent simply discovered it on its own.

Glossary of Neural Schema Terms

Action Node – A node that performs an action. *See also Primitive Action Node and Learned Action Node*

Action Link – A link that connects a schema node to its single action node.

Attention – One part of the mechanism's "consciousness" that has the task of determining which nodes are attended to on each given cycle. *See also "Consciousness"*

Bare Schema Node – A schema node that has only a link to its action and no context or result set.

"Consciousness" – In neural schema mechanism, this term is used to denote a specific implementation of Bernard Baar's psychological theory of human consciousness. It is comprised of two pieces (attention and the global broadcast) that, taken together, implement this theory. *See also "Conscious" Broadcast and Attention*

"Conscious" Broadcast – One part of the mechanism's "consciousness" that has the task of broadcasting a list of item nodes (determined by the attention component) to all schema nodes. *See also "Consciousness"*

Context Link – A link that designates an item node as being in the context set of a schema node. These links have an item node as input and a schema node as output.

Correlation – *See Schema Correlation*

Delegated Desirability – The desirability value that accrues in learned goal nodes over the course of a run.

Explicit Activation – The selection of a specific schema node whose action is to be performed.

Goal Node – A node that propagates desirability value in order to influence the agent's behavior to either pursue or avoid the environmental state that it designates. *See also Primitive Goal Node and Learned Goal Node*

Goal Link – A link that connects a goal node to an item node that embodies some or all of the environmental state that it designates.

Host Link – A link that connects a learned item node to one schema node whose context set designates part or all of the item node's verification conditions.

Host Schema Node – A schema node whose context set designates part or all of a learned item node's verification condition and to which it is connected via a host link. *See also Host Link*

Implicit Activation – A schema node is implicitly activation if it shares an action node with a different schema node that was explicitly chosen for activation.

Item Node – A node whose activation represents a state of the environment. *See also Primitive Item Node and Learned Item Node*

Learned Action Node – An action node that is created at runtime in response to a novel result spin-off of a schema node.

Learned Item Node – An item node that is created at runtime when a schema node is determined to be locally reliable but generally not reliable. The schema node in question becomes the learned item node's original host node. *See also Local Reliability, Reliability, and Host Schema Node*

Learned Goal Node – A goal node that is created at runtime in response to a novel result spin-off of a schema node.

Link – A connection between two nodes that could be of several specific types and may pass activation, desirability, and attentional value between the nodes.

Link Relevance – A measure of how relevant the linked item node (either input or output) is to the linked schema node.

Local Reliability – A measure of how likely a schema node is to be successful (its results occur in response to its action) if its action is performed within a short period of time of a previously successful activation.

Primitive Action Node – An action node that directly instigates an actuator and is present at design time.

Primitive Desirability – The desirability value that is given to primitive goal nodes at design time.

Primitive Item Node – An item node whose state is directly determined by sensory apparatus and is present at design time.

Primitive Goal Node – A goal node that generally represents an agent's drives and is present at design time.

Relevance – *See Link Relevance*

Reliability – A measure of the likelihood that a schema node's result will successfully obtain when its action is performed within the schema node's context.

Result Link – A link that designates an item node as being in the result set of a schema node. These links have a schema node as input and an item node as output.

Schema Correlation – A measure of how necessary a schema node's action is to achieving the schema node's result.

Schema Node – A node that embodies the heart of the explicit knowledge of the system. Along with its result set, context set, and action, a schema node describes a state of the environment that will result if its action is performed within a given context.

Spin-off – When a schema node creates a new version of itself with either a result link added or a context link added.

Synthetic Item Node – *See Learned Item Node*

Transition Correlation – A measure of how likely an item node is to change its state (turn *on* or *off*) when a given action is performed.

Type None Link – A generic link that does not pass activation or desirability, but gathers statistical data needed for learning.

References

Adams, S. (2001). *God's Debris: A Thought Experiment*, Kansas City, MO: Andrews McMeel Publishing.

Baars, B. (1988). *A Cognitive Theory of Consciousness*, Cambridge, MA: Cambridge University Press.

Baars, B. (1997). *In the Theater of Consciousness*, Oxford, UK: Oxford University Press.

Bickhard, M. H. (1997). Is Cognition an Autonomous Subsystem? In S. O'Nuallain, P. McKevitt, E. MacAogain (Eds.). *Two Sciences of Mind.* (115-131). John Benjamins.

Bogner, M. (1999). *Realizing 'consciousness' in software agents*. Unpublished doctoral dissertation, The University of Memphis, Memphis, TN.

Chalmers, D. (1990). Syntactic Transformations on Distributed Representations. *Connection Science, 2*, 53-62.

Damasio, A. R. (1994). *Descartes' Error*, New York: Gosset/Putnam Press.

Drescher, G. (1985). *The Schema Mechanism: A Conception of Constructivist Intelligence*. MIT M.S. thesis.

Drescher, G. (1987). A mechanism for early Piagetian learning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*. AAAI-1987.

Drescher, G. (1991). *Made Up Minds: A Constructivist Approach to Artificial Intelligence*, Cambridge, MA: MIT Press.

Fodor, J. (1975). *The Language of Thought*, Cambridge, MA: Harvard University Press.

Foner, L., & Maes, P. (1994, August). Paying Attention to What's Important: Using Focus of Attention to Improve Unsupervised Learning. In *Proceedings of the*

Third International Conference on Simulation of Adaptive Behaviour '94. Cambridge, MA: MIT Press.

Franklin, S. (1995). *Artificial Minds.* Cambridge, MA: MIT Press.

Franklin, S. (1997a). Global Workspace Agents. *Journal of Consciousness Studies*, 4 (4), 322-234.

Franklin, S. (1997b). Autonomous Agents as Embodied AI. *Cybernetics and Systems*, special issue on Epistemological Issues in Embodied AI. 28:6, 499-520.

Franklin, S., & Graesser, A. (1997). Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. *Intelligent Agents III*, Berlin: Springer Verlag, 21-35.

Franklin, S., Keleman, A., & McCauley, L. (1998). IDA: A Cognitive Agent Architecture. *Proc IEEE SMC'98*, 2636-7651.

Garzon, M., & Franklin, S. (1991). Neural Computability. In O. M. Omidvar (Ed.), *Progress In Neural Networks* (pp. 127 – 146). Norwood, NJ: Ablex.

Hebb, D.O. (1949). *The organization of behavior*. New York: Wiley.

Hill, W. (1985). *Learning: A survey of psychological interpretations*. (4th. Ed.). New York: Harper and Row.

Hopfield, J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the USA*, 9(2554).

Horgan, T., & Tienson, J. (1989, Spring). Representations Without Rules. *Philosophical Topics, 17*, 147-74.

Kanerva, P. (1988). *Sparse Distributed Memory*. Cambridge. Mass.:MIT Press.

Kintsch, W. (1988). The use of knowledge in discourse processing: A construction-integration model. *Psychological Review, 95*, 163-182.

Kintsch, W. (1998). Comprehension: a Paradigm for Cognition. Cambridge: Cambridge University Press.

Landauer, T. K., Laham, D., Rehder, B., & Schreiner, M. E., (1997). How well can passage meaning be derived without using word order? A comparison of Latent Semantic Analysis and humans. In M. G. Shafto & P. Langley (Eds.), *Proceedings of the 19th annual meeting of the Cognitive Science Society* (pp. 412-417). Mawhwah, NJ: Erlbaum.

Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of the acquisition, induction, and representation of knowledge. *Psychological Review,* **104**, 211-240.

Landauer, T. K., Foltz, P. W., & Laham, D. (1998). Introduction to Latent Semantic Analysis. *Discourse Processes,* **25**, 259-284.

Langley, P. (1987). A general theory of discrimination learning. In D. Klahr, P. Langley, & R. Neches (Eds.), *Production system models of learning and development*. Cambridge, MA: MIT Press.

LeDoux, J. E. (1989). Cognitive-Emotional Interactions in the Brain, In C. Izard editor, *Development of Emotion-Cognition Relations*, pages 267-290. Lawrence Erlbaum Associates, East Sussex, U.K.

Maes, P. (1989). How to Do the Right Thing. *Connection Science Journal, 1(3)*, 291-323.

Maes, P. (1992). Learning Behaviour Networks from Experience. In F. J. Varela & P. Bourgine (Eds.), *Toward a Practice of Autonomous Systems, Proceedings of the First European Conference on Artificial Life*. MIT Press/Bradford Books.

McCarthy, J., & Hayes, P. J. (1969). *Some Philosophical Problems from the Standpoint of Artificial Intelligence*. In B. Meltzer & D. Michie (eds.), Machine Intelligence (Vol. 4, pp. 463-502). Edinburgh: Edinburgh University Press.

Minsky, M. (1986). *The Society of Mind*. New York: Simon & Schuster.

Morrison, C., Oates, T., & King, G. W. (2001). Grounding the Unobservable in the Observable: The Role and Representation of Hidden State in Concept Formation and Refinement. *Working Notes of AAAI Spring Symposium Workshop: Learning Grounded Representations.*

Panksepp, J. (1995). The Emotional Brain and Biological Psychiatry. *Advances in Biological Psychiatry*, 1, 263-286.

Piaget, J. (1952) *The Origins of Intelligence in Children*. Norton, N.Y.

Piaget, J. (1954) *The Construction of Reality in the Child*. Ballentine, N.Y.

Pinker, S. (1995) Why the child holded the baby rabbits: A case study in language acquisition. In L. Gleitman & M. Liberman (Eds.), *Invitation to Cognitive Science, 2nd Edition*. Volume 1: Language. Cambridge, MA: MIT Press.

Ramamurthy, U., Bogner, M., & Franklin, S. (1998). Conscious Learning In An Adaptive Software Agent. In *Proceedings of The Second Asia Pacific Conference on Simulated Evolution and Learning* (pp. 24-27). Canberra, Australia.

Rescorla, R. A., & Wagner, A. R. (1972). A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In A. H. Black & W.

F. Prokasy (Eds), *Classical conditioning II: Current research and theory* (pp. 64-99). New York: Appleton-Century-Crofts.

Russell, S., & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*, Upper Saddle River, NJ: Prentice-Hall, Inc.

Schank, R. C., & Abelson, R. P. (1977). *Scripts, plans, goals, and understanding*. Hillsdale, NJ: Erlbaum.

Spence, K. (1936). The nature of discrimination learning in animals. *Psychological Review*, 43, 427-449.

Sun, R., Peterson, T., & Merrill, E. (1999). A hybrid architecture for situated learning of reactive sequential decision making. *Applied Intelligence*, Vol.11, pp.109-127.

Sun, R., Peterson, T., & Merrill, E. (2001). From implicit skills to explicit knowledge: a bottom-up model of skill learning. *Cognitive Science*, Vol.25, No.2, pp.203-244.

Appendixes

Appendix A – Examples of Schema Nodes

This appendix lists examples of schema nodes created during one run of neural schema mechanism in the Wumpus World environment. The nodes are divided into several different categories that represent common types produced in numerous runs. The node listings are modified forms of the entries into the knowledge log file for a particular run.

Schema Nodes for Positional Knowledge

The most common type of sense represented as item nodes is the positional sense. These are item nodes whose state reflects the grid position of the agent. The agent can only be in one grid position at a time. Baring any obstacles, movement actions taken by the agent will transition two positional item nodes, one that represents the position that the agent is in at the time the action is taken and one that represents the new position to which the agent moves. Within this category there are primarily two different types of schema nodes that get created aside from the original result spin-offs. One of these types references the same item node in both the context and result sets. These types of schema nodes seem trivial but represent a necessary piece of knowledge that the agent does not know initially, namely that moving from a position means that the agent is no longer in this position. These types of schema nodes directly address the Frame Problem (McCarthy & Hayes, 1969). For the agent there is more than just this simple fact being addressed, there is also the fact that not all of the agent's actions are movement actions and these types of schema nodes are part of the process of this discovery. Also, not all movement actions in all positions result in a transition to a new position; the agent must learn which states those are.

```
schemaNodeMoveSouth.0
    action:
        moveSouthActionNode
    context:
    result:
        sensePos20 = off

schemaNodeMoveSouth.1.0
    action:
        moveSouthActionNode
    context:
        sensePos12 = on
    result:
        sensePos12 = off

schemaNodeMoveWest.0
    action:
        moveWestActionNode
    context:
    result:
        sensePos30 = on

schemaNodeMoveNorth.0
    action:
        moveNorthActionNode
    context:
    result:
        sensePos20 = on

schemaNodeMoveSouth.0.0
    action:
        moveSouthActionNode
    context:
        sensePos20 = on
    result:
        sensePos20 = off

schemaNodeMoveWest.0.0
    action:
        moveWestActionNode
    context:
        sensePos31 = on
    result:
        sensePos30 = on
```

```
schemaNodeMoveNorth.0.0
    action:
        moveNorthActionNode
    context:
        sensePos30 = on
    result:
        sensePos20 = on

schemaNodeMoveWest.2.0
    action:
        moveWestActionNode
    context:
        sensePos21 = on
    result:
        sensePos20 = on
```

Schema Nodes for Non-Positional Knowledge

Of course not all knowledge in the system is positional. Listed here are some schema nodes that relate various types of actions to non-positional item nodes.

```
schemaNodeMoveNorth.1
    action:
        moveNorthActionNode
    context:
    result:
        senseBreezeNode = on

schemaNodeMoveWest.4
    action:
        moveWestActionNode
    context:
    result:
        senseEatenNode = on

schemaNodeGrab.0
    action:
        grabGoldActionNode
    context:
    result:
        senseGotGoldNode = on
```

```
schemaNodeMoveNorth.1.1
    action:
        moveNorthActionNode
    context:
        senseGotGoldNode = on
    result:
        senseBreezeNode = on

schemaNodeMoveWest.4.0
    action:
        moveWestActionNode
    context:
        senseStenchNode = on
    result:
        senseEatenNode = on

schemaNodeMoveNorth.5.0
    action:
        moveNorthActionNode
    context:
        senseStenchNode = on
    result:
        senseEatenNode = on

schemaNodeMoveSouth.6
    action:
        moveSouthActionNode
    context:
    result:
        senseBumpSouthNode = on
```

Schema Nodes for Positional and Non-Positional coordination

Eventually, the schema nodes must make some correlation between positional item nodes and non-positional item nodes. These can become some of the most useful schema nodes since they represent knowledge that restrains the instantiation of certain senses to specific places in the environment. Alternatively, they can limit the non-positional results of actions to being dependent on the agent's location. For example, the ability to get the gold is restrained to a particular spot.

```
schemaNodeMoveNorth.1.0
    action:
        moveNorthActionNode
    context:
        sensePos20 = on
    result:
        senseBreezeNode = on

schemaNodeGrab.0.0
    action:
        grabGoldActionNode
    context:
        sensePos03 = on
    result:
        senseGotGoldNode = on

schemaNodeMoveEast.4.0
    action:
        moveEastActionNode
    context:
        senseGlimmerNode = on
    result:
        sensePos03 = on

schemaNodeMoveNorth.2.2
    action:
        moveNorthActionNode
    context:
        sensePos12 = on
    result:
        senseStenchNode = on

schemaNodeMoveSouth.4.0
    action:
        moveSouthActionNode
    context:
        senseBreezeNode = on
    result:
        sensePos20 = on
```

```
schemaNodeMoveNorth.6.1
    action:
        moveNorthActionNode
    context:
        senseGlimmerNode = on
    result:
        sensePos03 = on
```

Schema Nodes with Multiple Contexts or Results

Over time, the agent will learn that there may be several item nodes that transition their states in response to an action or that there are multiple item nodes whose state contributes to the reliability of a schema node. The result of this is the creation of schema nodes that have more than one item node in the context or result sets.

```
schemaNodeMoveSouth.3.1.1
    action:
        moveSouthActionNode
    context:
        senseStenchNode = on
        sensePos11 = on
    result:
        senseStenchNode = off

schemaNodeMoveNorth.2.3.0
    action:
        moveNorthActionNode
    context:
        sensePos21 = on
    result:
        senseStenchNode = on
        sensePos11 = on

schemaNodeMoveWest.4.0.0
    action:
        moveWestActionNode
    context:
        senseGotGoldNode = on
        senseStenchNode = on
    result:
        senseEatenNode = on
```

```
schemaNodeMoveSouth.3.1.2
    action:
        moveSouthActionNode
    context:
        senseStenchNode = on
        sensePos02 = on
    result:
        senseStenchNode = off

schemaNodeMoveWest.4.0.1
    action:
        moveWestActionNode
    context:
        senseStenchNode = on
        sensePos02 = on
    result:
        senseEatenNode = on

schemaNodeMoveSouth.4.0.0
    action:
        moveSouthActionNode
    context:
        senseBreezeNode = on
        sensePos10 = on
    result:
        sensePos20 = on
```

Schema Nodes Using Learned Action Nodes

Here is an example of a learned action node being used to learn a higher-level task. In this case `ActionSchemaNode120` is connected to the learned goal node that represents the transitioning of the `senseGotGoldNode` to the *on* state. You will notice that the results of these more abstract actions can span multiple positions or other sensory item nodes. The reason for this is that the abstract action that might be called "Get Gold" might have different results for different situations even though the overall action is the same. In this example we see the result set of `senseGotGoldNode` for

one schema node and `sensePos03` for the second schema node. The transition of these two item nodes would come in different steps in a chain of schemas that would result in the attaining of the "Get Gold" action (or goal).

```
SchemaNode120.0
    action:
        ActionSchemaNode120
    context:
    result:
        senseGotGoldNode = on

SchemaNode120.1
    action:
        ActionSchemaNode120
    context:
    result:
        sensePos03 = on
```

Schema Nodes Using Learned Item Nodes

The learned item nodes (those with a capital "I" in the version portion of the node name) shown in the contexts of these schema nodes demonstrate the use of synthetic item nodes in schema nodes. The `schemaNodeMoveWest.4I.1` generally represents that the agent is next to the wumpus and the `schemaNodeMoveSouth.3I.1` generally represents that the agent is within a square of the gold.

```
schemaNodeMoveWest.1.0.1
    action:
        moveWestActionNode
    context:
        senseStenchNode = on
        schemaNodeMoveWest.4I.1 = on
    result:
        senseEatenNode = on
```

```
schemaNodeMoveWest.5.0.1
    action:
        moveWestActionNode
    context:
        senseStenchNode = on
        schemaNodeMoveWest.4I.1 = on
    result:
        sensePos01 = on

schemaNodeMoveEast.6.0
    action:
        moveEastActionNode
    context:
        schemaNodeMoveSouth.3I.1 = on
    result:
        sensePos03 = on
```

Appendix B – Algorithm Flow for the Node Manager

Part of the reason why the current implementation is still computationally expensive is the fact that it is simulating parallelism on a single processor system and uses a node manager to maintain synchrony. Shown here are the steps taken by the node manager on every cycle. Note that a large quantity of these steps requires a sequential traversal of all of the nodes in the system as denoted by the "*". An analysis has, as of yet, not been done to determine if some of these loops could be combined without affecting the integrity of the mechanism.

a. *Fire – Each node's "fire" method is run. The "fire" method determines if the node's activation is high enough to be passed and, if so, transfers the appropriate amount of activation to its links. It also performs some maintenance tasks internal to the node such as adjusting time values.
b. *Check for result and context spin-offs – In addition this also checks for the possibility of synthetic item node creation.
c. *Store link states – All of a node's input links are told to lock into a temporary location the current state of their input node. This will be compared to any changes that occur as a result of actions taken during this cycle.
d. *Decay output links – This tells each node to perform a decay operation on the weights of their output links. Since every link is an output link for one node, this effectively decays all links without decaying any link twice.

e. *Pass desirability value – The desirability value in each node is passed to the appropriate links. The links will not actually pass the desirability value on until the following step.

f. *Receive desirability value – Each node retrieves the desirability value from each of its links and calculates its new desirability value.

g. *Pass attentional value – Similar to the "Pass resistance" step, this step transfers the current degree of attention to the links who will hold it until the next step.

h. *Receive attentional value – Each node retrieves the attentional value being sent to it through its links and calculates its new attentional value.

i. *Normalize attentional value – The total amount of attention in the system remains constant. This step adjusts the attentional value of each node to the appropriate normalized level.

j. Update "conscious" links – The list of item nodes to be broadcast is created at this point and links are created, if they do not already exist, between nodes that are "conscious."

k. *Update gain value – This step updates the current gain value based on the current state of the network.

l. Update delegated desirability value – This instigates the recalculation of the delegated desirability value in all of the learned goal nodes based on the current state of the network.

m. Execute Action – This is where the system determines if an action is going to be taken or not and, if so, performs that action.

n. *Broadcast – The current list (created in step "j") of item nodes to be "consciously" broadcast is sent out to all schema nodes.

o. *Normalize activation values – If the total activation in all of the nodes of the network gets above a certain value, then the activation in every node is adjusted so that the total activation is ten percent below the threshold level.

p. Update registered elements – Any new nodes that were created in step "b" are officially added to the network. This delay avoids unnecessary work in the steps that have occurred since the nodes were created. It also avoids miscalculations based on the number of nodes in the network.

q. Update the user interface display – The user interface is updated to reflect all of the changes for the current cycle. This may or may not require a full traversal of the nodes in the network depending on the display mode.

Appendix C – Author's publications

Learning Motivational Structures Using Neural Schemas, Cognitive Science Quarterly (currently in review).

Neural Schemas: A Mechanism for Autonomous Action Selection and Dynamic Motivation, 3rd WSES Neural Networks and Applications Conference, Switzerland, 2002.

An Emotion-Based "Conscious" Software Agent Architecture, McCauley, Franklin, and Bogner, Proceedings of the International Workshop on Affect in Interaction, Siena, Italy, October 1999, published as a book chapter in "Affective Interactions", Ed. A. Paiva, Lecture Notes on Artificial Intelligence, LNAI 1814, Springer, 2000.

Implementing Emotions in Autonomous Agents, Master's Thesis, The University of Memphis, February 1999.

Delivering Smooth Tutorial Dialog Using a Talking Head, McCauley, Gholson, Hu, Graesser, and The Tutoring Research Group, Presented at the Workshop on Embodied Conversational Characters, Lake Tahoe, CA, September 1998.

An Architecture for Emotions, McCauley and Franklin, Presented at the American Association for Artificial Intelligence's fall symposium, "Emotional and Intelligent: the Tangled Knot of Cognition," Orlando, FL, October 1998.

Index