A (Somewhat) New Solution to the Binding Problem

Leon Barrett, Jerome Feldman, and Liam Mac Dermed

April, 2006

Abstract

To perform automatic, unconscious inference, the human brain must solve the "binding problem" by correctly grouping properties with objects. We propose a connectionist, localist model that uses short signatures, rather than temporal synchrony or other means, to do this binding. The proposed system models our ability to both perform unification and handle multiple instantiations of logical terms, among other things. To verify its feasibility, we simulate our model with a computer program modeling simple, neuron-like computations.

1 Introduction

How do humans use their complicated, neural brains to think? Though many have considered the problem, it is by no means solved. For instance there are several variants of the "binding problem," which asks how a massively parallel system can achieve coherence. The most striking examples involve subjective experience and therefore remain intractable to experimentation. For example, we know that visual processing involves dozens of separate brain areas, yet we perceive the world as a coherent whole. The mechanism for this is still not well-understood. Even leaving subjective experience aside, there are still compelling technical problems in understanding how a neural network can solve crucial computational problems, such as those that arise in reasoning about and acting in the world.

A basic problem, and the one that we will focus on, is the "variable binding" problem. As a first example, consider your ability to pick up objects. Depending on the object, its current position, and your goals, you have a very wide range of ways of grasping and manipulating the object, all realized by the network of neurons that is your brain. This is an instance of the variable binding problem because your choice of values for the three variables *object*, *position*, and *goal* has consequences throughout your brain on how the action is carried out. In conventional computing, we assume that different program modules all have access to the values of (global) variables and can modify their behavior appropriately. Similarly, any theory of neural computation needs some mechanism for achieving this kind of global effect.

In the domains of problem solving, language understanding, and other symbolic behavior, the variable binding task becomes even more complex and interesting. The linguist Ray Jackendoff has suggested that the variable binding problem is the key to any neural theory of language [Jackendoff, 2002]. Both language and problem solving involve operations on variables before they have values assigned to them. For example, humans routinely use rules of logic like "All humans are mortal," equivalently written *Human(X)* $\rightarrow$ *Mortal(X)*, to perform similar inferences about many different people. An obvious such use of variables in language understanding is reference resolution, which has several forms. The clearest case is the task of binding pronouns to their referred objects, but there are many other examples such as, "the winner of the 2012 U.S. Presidential election," which has meaning even if no one yet knows precisely who that person is. Humans can perform this inference effectively with a moderate number of variables (famously, humans can deal with about 7 ± 2 objects) without explicit effort, and we would like to model how this might work in a massively parallel neural system.

Several possible models of how the brain performs automatic inference already exist; they range from holistic to localist and have various degrees of support. In this paper, we propose a highly localist inference model based on SHRUTI [Shastri, 1999] that uses short signatures rather than temporal synchrony; such a strategy has been suggested by Browne and Sun [2000]. This model is supported by an implemented computer simulation using mostly neuron-like units and simple, biologically plausible signals. Not all aspects of the model were implemented in a biologically plausible way; in particular, some of the more complicated interconnections were done using digital logic rather than by emulating neuron-like units. For more on this implementation, see §5.

2 Background

The variable binding problem is well known in connectionist and neural computation circles. There have been ingenious solutions proposed over the years, and our proposal depends crucially upon them. However, no previous designs have been both computationally complete and consistent with biological findings.

2.1 A Brief Survey of Solutions to the Variable Binding Problem

Researchers attempting to construct working models of the mind have developed a variety of mechanisms to solve the binding problem. Of these, the localist attempts have proven to be more useful (to researchers, at least) than their distributed counterparts. In the past two decades, three main models for performing variable binding have been used by localist systems. These three models can all be defended as generally neurologically plausible.

The first localist systems, from about two decades ago, used brute force enumeration of all possible bindings. It is evident that this approach is unmanageable with anything more than a few variables, and was never taken as realistic solution, but rather a work-around to allow research into other aspects of connectionist networks. Recently, however, O'Reilly et al. [2001] present a means of using coarse-coded conjunctive binding to mitigate the exponential complexity of full enumeration. Two examples of systems that use enumeration are described below.

- *Parallel Relaxation Algorithm* – [Ballard, 1986, Lima, 1992]
  "All possible variable substitutions were prewired using hand-coded weights, with incompatible bindings being represented by units being connected with inhibitory links." [Browne and Sun, 2001]
- *Cortical coarse-coded conjunctive binding* – [O'Reilly et al., 2001]
  "We think the cortex constructs low-order conjunctions using coarse-coded distributed representations to avoid the combinatorial explosion usually associated with conjunctive solutions to the binding problem."
- More generally, representations based on structured combinations of activation, like coarse-coding, are among the most common and best attested connectionist modeling techniques. They play an important role in binding models, including ours, but do not suffice for a total solution.

A second approach has been to use sign (signature) propagation. In sign propagation, each variable has its own node (a group of neurons working together). This node then takes on a particular signature that corresponds to a concept, so the signature is essentially treated as a name for the object. Signatures can be propagated from one node to the next when unification occurs. Various different methods of propagation and creating signs have been developed including amplitude modulation (where the magnitude of activation corresponds to a concept) and frequency modulation (where the rate of neural firing corresponds to a concept). Two examples of this type of network are:

- *ROBIN – The Role Binding and Inferencing Network* – [Lange and Dyer, 1989, Sun, 1989, 1992]
  "Each constant has a unit in the network that has a uniquely identifiable sign, bindings are formed by passing these signs around the network. A dynamic binding exists when the binding node for a variable has an activation matching the bound constant's sign. These bindings are propagated in parallel and are determined by a constraint satisfaction process, performed using hand coded connection weights."[Browne and Sun, 2000]

- *QNET* – [Looke, 1995]
  "Performs variable binding using a localist representation where individual units in the network are characterized by amplitudes in more than one frequency. . . . only achieves pattern matching, not full unification" [Browne and Sun, 2000]

The third and most widespread approach is that of phase synchronization, also known as temporal synchrony. This approach breaks the cycle of neural firing into discrete time slices. When a variable node fires in-phase with a concept node, this represents a binding between those nodes. For example, if the node representing "Fido" is firing at the same time as the node for "dog," then this indicates that "Fido" has the "dog" property. Although this approach is quite attractive for many reasons, it also has some shortcomings. A computational problem is the loss of speed due to the need of repeated iterations to get nodes firing in phase, as well as the limited number of phases (thus a limited number of bindings) that can be reasonably supported. Also, there has been a dearth of evidence for the temporal synchrony model. That is not to say that there is significant evidence against it, but the fine-grained synchrony described in these models has not yet been observed. There is one localist system in particular using temporal synchrony that is particularly mature.

- *SHRUTI* – (Lokendra Shastri - www.icsi.berkeley.edu/∼shastri/shruti/)
  Shruti is the leading localist representation to use phase synchronization. Over the years it has adopted and incorporated the improvements of many other projects. "These projects as well as other approaches using temporal synchrony to solve problems in perceptual processing have been developed (e.g., Strong and Whitehead, 1989; Wang, Buhmann, and von der Malsburg, 1990; Horn and Usher, 1991; Grossberg and Somers, 1992; Hummel and Biederman, 1992; Niebur, and Koch, 1994)." [Shastri, 1999]

2.2 How we frame the variable binding problem

Essentially all recent connectionist models of the variable binding problem frame the problem in the same way. There are collections of facts and of rules of inference, expressed like those of traditional AI logical reasoning systems. The basic mode of operation is to pose a query as an assertion (usually with some unbound variables) and have the model return one or more correct bindings. Although a separate ontology is not strictly necessary, most systems, including ours, include one as well. All of the models we are discussing perform restricted inference, intending to model human automatic, unconscious reasoning. Some recent models [Wendelken and Shastri, 2000] even include some probabilistic inference. There is currently no complete, biologically plausible model of human Bayesian or other probabilistic reasoning, although "neuroeconomics" is a hot topic in several fields.

3 Our approach

Our model of a neural binding process uses ideas from both the signature and temporal synchrony approaches. It seems unbiological to assume that a network of neurons can reliably pass around enough signature information to uniquely identify any possible concept (∼20 bits). The key to our solution relies on the standard programmer's tool of indirect addressing. We know that people can only deal with a limited number of variables (∼8) at a time. Suppose that we only require the neural network to manipulate patterns that determine which of some 8 entities is involved in the current reasoning process (∼3 bits). In this case, then we can use many of the same structures pioneered by the temporal synchrony approaches. Such a model is a straightforward extension of SHRUTI, and is mentioned as such by Browne and Sun [2000]. In fact, the implemented computational model of SHRUTI actually uses such short signatures rather than simulate synchronous signals. However, instead of having the presence of a signature in an entity structure indicate that the signature is bound to the entity, as in SHRUTI's temporal synchrony model, we use a centralized structure to concretely represent such a mapping. This allows for unification of signatures that have been determined to represent the same object.

Loosely following the usage in conventional logic, we will call these ∼8 short signatures **fluents**. Each fluent represents a single object or concept; it is linked to something familiar by a central structure mapping it to some object (concept) in the ontology of our model system. As a simple example, imagine a query *?Mortal(Clinton)* (that is, is Clinton mortal?) in a neural system that had the knowledge:

1. *Human(X)* → *Mortal(X)*
2. *Human(Clinton)*

Each logical predicate and implication is given a neural structure to represent it. Ignoring the details for now, our system would first associate *Clinton* with some fluent, say *f3*. A structure representing *Mortal(X)* is linked to all rules, like (1), that have it as a consequent. So, our query would activate rule (1) with the binding of *X* to *f3*. That is, *?Mortal(f3)* will cause queries to all rules and facts to seek supporting or contradictory evidence, including (1). Now rule (1) has *Human(X)* as its antecedent so it will in turn activate relevant rules and facts about *?Human(f3)*. One recipient of this query is fact (2). Because *f3* is bound to *Clinton*, fact (2) entails positive evidence for the query. So, fact (2) sends a + signal to rule (1) which can then send a + signal to *Mortal(X)*. Thus, question signals move in one direction, and then answer signals retrace the path back. (We will look later at the case where the query involves finding bindings as well as + or - answers.)

3.1 Subtasks

Even the simple introductory example presented above requires solving a number of problems in neural computation, and each problem might be approached in various ways. We have implemented a particular system, with particular choices of design, to test and illustrate the ideas, but we will describe a range of possible designs that might be of interest. Of course, what we really want is not just a means for doing computation, but also a model of how our brains actually work, so we must describe a range of computationally and biologically plausible solutions to these problems that can be sorted out by means of experiment. For convenience, we will name and label various tasks and proposed models of how they might work. Table 1 summarizes the results that will be presented, with each solution laid out in detail in §4.

Problem:	**Approach:**	**Solutions:**
A. initial setup	link input item to fluent	A1 ∼ X-bar; A2 ∼ address
B. propagate query	*?* node activation w/ fluents	B1 ∼ basic
C. get ± answer	return a single + or -	C1 ∼ basic
D. get item answer	bind each open fluent	D1 ∼ base, D2 ∼ unification
E. clusters	copies of logic terms	E1 ∼ extends B1, B2
F. use ontology	taxonomic inferences	B2 ∼ with ontology
G. multiple answers	return multiple bindings	G1

Table 1: Subtasks and Possible Solutions

Our initial example required only subtasks A-C: binding fluents to objects, propagating queries, and sending evidence answering these queries. However, many queries ask more than just whether something is true;

instead, they also require one or more answer items. For example, a query about the color of grass might be posed as *?Color(grass, X)*, where X denotes a variable which should hold the answer(s). Getting this answer is task D, and here it involves binding one fluent to the fixed input "grass" and choosing another "open" fluent to hold the answer. (It is much more difficult to return multiple sets of answers, and that is task G.)

Subtask E involves an internal issue. Our binding model relies on the parallel activation of terms in many logical relations and it is important that these not get confused. For example, we might need multiple instances of the *Owns* relation for use in rules like:

$$Owns(X,M)\ \&\ Owns(Y,N) \rightarrow CanSwap(X,Y,M,N)$$

The system needs to keep track of several *Own* relationships and this requires multiple copies. There are obvious limits to how many such copies we can keep active and this fact leads to solutions such as E1.

Subtask F involves combining our inference schemes with a neurally plausible implementation of a taxonomic lattice or ontology. Inference is technically possible to do without such an ontology; we could always include rules like: *canary(X)* $\rightarrow$ *bird (X)* and facts like *canary(Tweety)*. However, ontologies are efficient and plausible knowledge stores, and we should be able to model how they could work with our rules and mechanisms.

Finally, subtask G is the problem of returning multiple answer items. For example, imagine listing all the books you know by Stephen King. This is substantially harder than getting a single answer, and we propose solution G1 as a method for this.

3.2 Computation-Level Description of the Fluent Binding Model

The general structure of our model is highly influenced by SHRUTI [Shastri, 1999]. In particular, the mechanisms for storing short-term relational knowledge, such as *Own(John, Book117)*, and computing implications, such as *Give(X,Y,M)* $\rightarrow$ *Own(Y,M)*, are almost identical to SHRUTI's—except for the treatment of dynamic binding, which is the main contribution of this paper.

The smallest structures in our model are **nodes**. The basic idea is that these are clusters of between several and hundreds of neurons, and these clusters behave as simple computational units. Connections between them are used to perform calculations; for example, a node could hold activity representing a belief value, and connections between three nodes could cause one of them to represent the boolean "and" value of the other two. We build our entire network out of these simple primitives.

Using combinations of simple structures, our system represents short-term beliefs about the world. If we are told, "Mary owns book117," our system records this information as activation in a structure called a **predicate**. (SHRUTI calls predicates, as well as mediators [described below], **focal clusters**.) Each predicate has a semantic meaning (e.g., the predicate in Figure 1 represents an instance of the relation *Give*), and it has one or more **roles** (a type of node), which can be filled by fluents representing entities (more on that later). Entities are people, places, or things, that are either specific instances (such as "Bill Gates"), or general categories (such as "Man"). How these entities are represented to fill these roles will be described later in more detail. The idea is that when a predicate (such as *Give*) holds roles (such as *giver=Mary*, *recipient=John*, *object=book117*), then the predicate has a semantic meaning (such as *Give(Mary,John,book117)*, which is the same as "Mary gave John book117").

A predicate also has a pair of **collector** nodes, + and -, which hold the current truth value of the relation. If the + node is active, then the relation is believed to be true, and if the - node is active, then the relation is believed to be false. These values need not be all-or-nothing; they can represent a whole range of values. As an example, a high activation of the + node in the *Give* predicate with bindings *(Mary,John,book117)* would correspond to the belief Mary gave book117 to John. If, instead, the - node was active, then this would represent a belief that Mary did *not* give book117 to John. The + and - nodes are mutually inhibitory, so positive and negative evidence compete. If neither node wins, and both + and - nodes are active, the predicate indicates uncertainty.

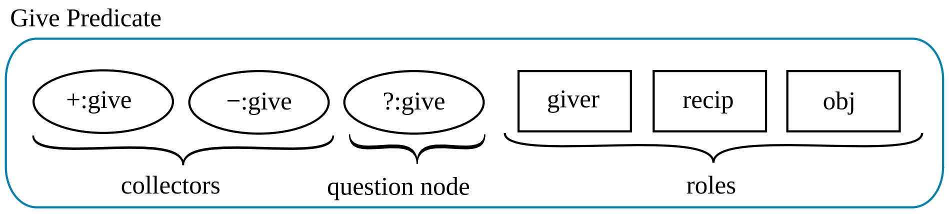

Figure 1: Give Predicate

In our model, as in Shruti, the predicate also has a **question node**, *?*, which is activated when the current set of bindings is being queried. If the *?* node is active, then we are wondering about the truth value of the predicate. This causes the system to actively seek out long-term knowledge and implications of this relation.

A predicate connects to other structures in the following reciprocating ways: Activation of the *?* node will trigger questioning of facts and implications that may support or refute the truth of the predicate. Structures whose + or - node is activated by this questioning (indicating that they have relevant information) will then activate the + or - node of the inquiring predicate. So, the question *?Give(Mary,John,book117)* (Did Mary give John book117?) will yield the answer *+Give(Mary,John,book117)* (Yes, Mary gave John book117.) if there is evidence to support this.

To store long-term knowledge, we use a **fact** structure. Each fact is associated with a particular predicate. It represents a mapping from a specific set of bindings of that predicate to a truth value. When that predicate's question node is activated, activation spreads to the question node of all related fact clusters. If the bindings match, then the fact sends a signal to the predicate. The signal's strength is determined by the weight of connections to the predicate's collectors, representing the stored long-term truth value. For example, if we have the fact "John owns book 117," when we pose the question "Does John own book 117?" (Figure 2) the corresponding fact will activate the query's plus collector, verifying that John does indeed own book 117. On the face of it, facts may not seem necessary, as this same sort of behavior could be achieved by connecting predicates directly to each other. However, facts will be needed once we add an ontology, so we introduce them now.

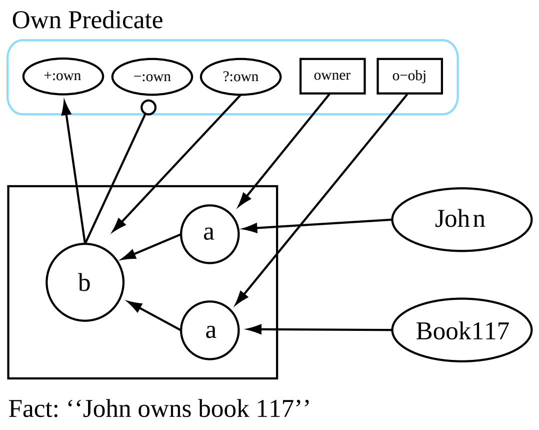

Figure 2: Fact representing "John owns book 117"

Given this store of knowledge, we need a way of using it to infer additional information. Supposed we know that if person X gives item A to person Y, then person Y owns item A. (i.e., *Give*$(X,Y,A) \rightarrow$ *Own*(Y,A)). At first we may think that a direct connection would be an acceptable solution (Figure 3). However, after a quick reflection it should become clear that there is more to inference than simple transmission of activation. Additional mechanisms are needed for multiple antecedents, role consistency, and default role filling. These functions are handled by a **mediator** cluster.

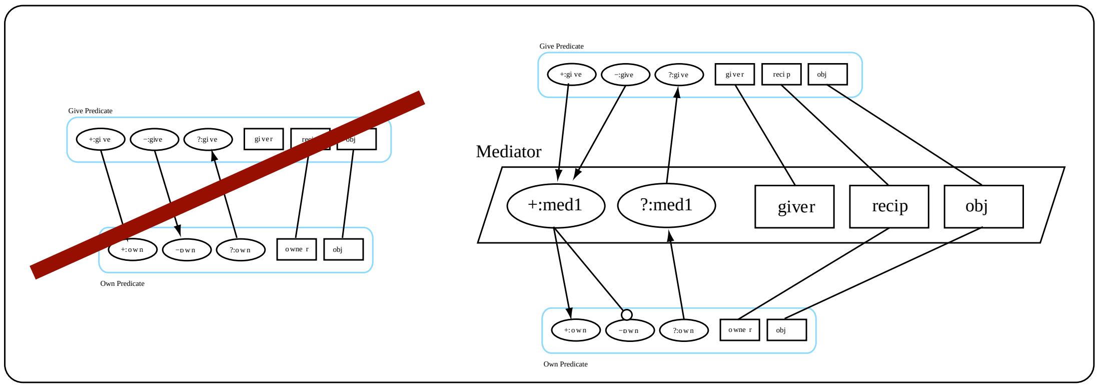

Figure 3: A naive approach to inference (left). A more effective idea (right).

A mediator conjunctively connects one or more antecedent predicates to a single consequent predicate. It has a role for every variable used by the antecedents or consequent, a single collector *+*, and a query node *?*. The collector is linked to the positive or negative collector of each antecedent, so that it will be active if the antecedents are active, thus performing a conjunction over evidence. It also has a connection to the positive or negative collector of the consequent so that (positive or negative) conclusions will follow. Also, when a predicate *?* node is active, bindings are passed from the consequent to the mediator and then on to the antecedents, so that questions about objects are passed onwards. Similarly, belief about the antecedents spreads through the mediator to the consequent via the collectors. Because mediators can connect to either the *+* or *-* nodes, implications can include negation in any antecedent or consequent.

So, one can see predicates and mediators as performing different tasks. Predicates have semantic meaning and perform "or" operations over mediators, with possible inversions caused by negative evidential weights. Mediators, on the other hand, represent implications and perform "and" operations over predicates, again with possible inversions. Thus, the basic logical operations (and, or, not) are possible in inference.

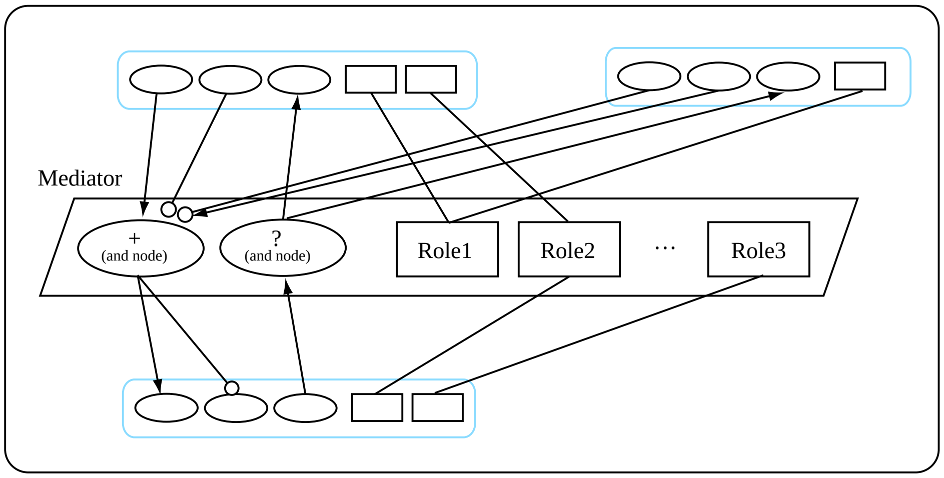

Figure 4: Detail of a Mediator

So far, we have treated roles as being magically filled by fluents that represent entities. This obviously needs to be refined; we want to have each entity represented by a fluent that serves as a "pointer" to the entity in the ontology. However, it is also impractical to have the full description of each entity filling a role in a predicate or mediator; this is essentially the "signature" approach of Lange and Dyer [1989] and requires coherent patterns of some 20 bits for each entity.

In our model, the roles in predicates and mediators are represented by small banks of units (∼3) that hold an activation pattern (see Figure 5). A central structure then links this to the role's entity. This design is an indirect addressing scheme and each of the ∼8 registers is called a **fluent**. A fluent is dynamically bound to the entity it represents. The key idea is that inference now involves only passing and comparing a fluent's (3 bit) address, and the effect is similar to transmitting the full (∼20 bit) address of the entity.

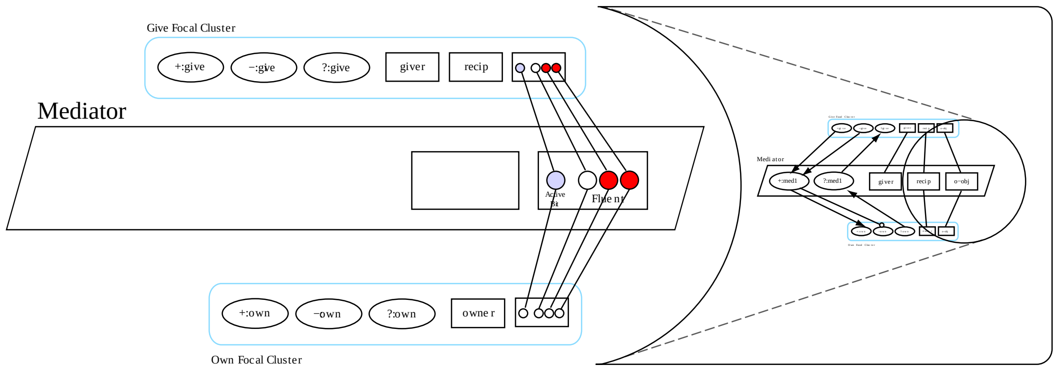

Figure 5: Details of a fluent role

The assignment and maintenance of the connection between fluents and their entities occurs in a structure called the **fluent binder**. Roles that require the use of a new entity request an unbound fluent from the fluent binder. This fluent then becomes bound to the new entity by means of either an address or a dynamic binding network; this is Task A in Table 1, and these two alternatives are examined more closely in §4.1. A fluent number is thus much like a time slice in SHRUTI. However, a fluent not only indicates that a role is bound to an object, but also provides a link so that the original object can be activated on demand. It is this direct link that makes fluents more useful than time slices. Beyond maintaining the link between entity and fluent, the fluent binder also has other important functions that will become clear as the details are more closely examined in later sections.

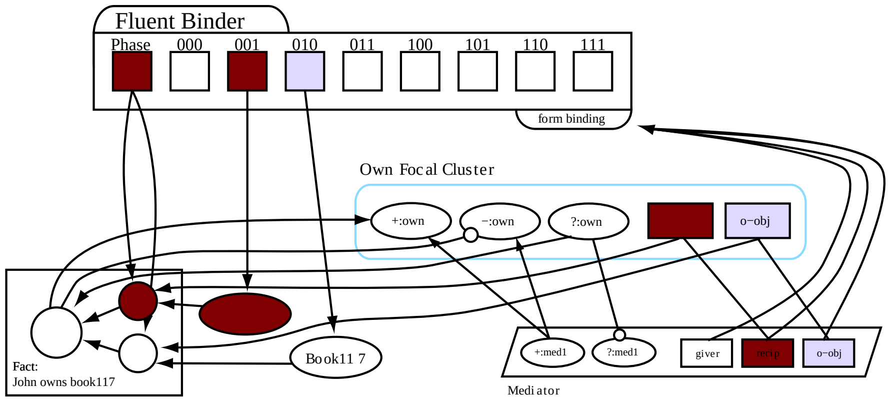

Figure 6: The fluent binder and its connections

This is the basic design of the inference network, and it allows the inference model to produce answers to all sorts of interesting questions. Consider the example network of Figure 7 (left). Here, we have several predicates, including *Scary*, *Big*, and *Fido*. There are also several mediators, such as the one connecting *Scary* to *CanBite* and *Big*. Finally, we also have two facts, indicating that dogs can bite and that Spot is big.

Spot(x) $\rightarrow$ *Dog(x)*
Fido(x) $\rightarrow$ *Dog(x)*
Fact: *Big(Fido)*
Fact: *CanBite(Dog)*
Big(x) & CanBite(x) $\rightarrow$ *Scary(x)*

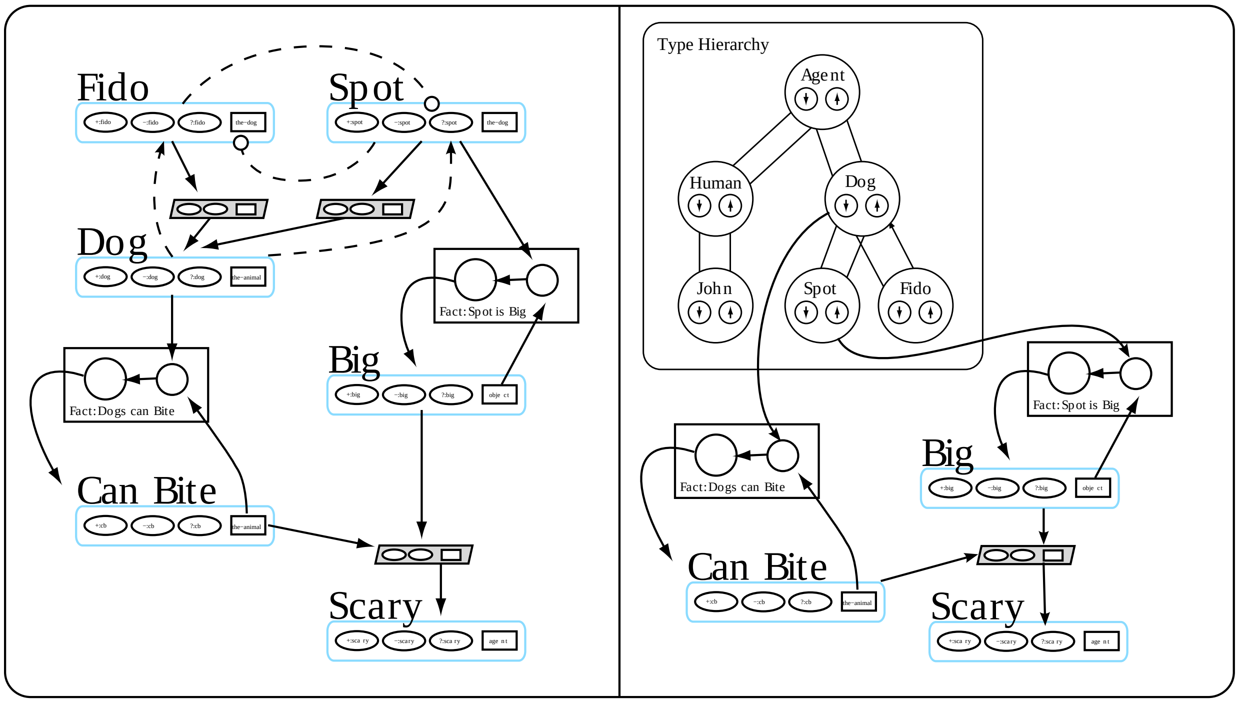

Figure 7: Simple design without use an ontology (left). A more sophisticated design with an ontology (right).

If we want to ask "Is Spot scary?" we activate *?Scary(f1)* with fluent *f1* bound to *Spot*. Question signals travel up the network through predicates and mediators. The fluent binder and fluent numbers will activate the proper facts, providing evidence for predicates. Then, these answers will travel back down, resulting in positive activation arriving at *Scary(X)*, indicating that Spot is indeed scary.

Similarly, to ask "Is Fido scary?" we activate *?Scary(f1)* with *f1* bound to *Fido*. In this case, we have no fact about Fido's size, so no activation is provided to *Big*. So, no activation is provided to the mediator below *Big* or to *Scary*, and we cannot conclude anything about Fido's scariness. In this way, a lack of knowledge can be represented, removing the need for the closed world assumption.

There are some limitations of the system as described so far. For example, if we try to ask "Is there a dog that is scary?" by activating *?Scary(Y)* and *+Dog(Y)*, Fido will never receive activation. One (inferior) way to solve this problem is to add some additional implications (shown with dashed arrows). These implications are contradictory, but this is manageable in the described system because the activations sent are real-valued belief values which can deal with contradictory evidence.

$$Dog(X) \rightarrow Fido(X)$$

$$Dog(X) \rightarrow Spot(X)$$

$$Spot(X) \rightarrow \neg\ Fido(X)$$

$$Fido(X) \rightarrow \neg\ Spot(X)$$

These implications will partially activate both *Fido* and *Spot* when *Dog* is activated. Unfortunately, this creates the implication that if X is a dog, and X is not Spot then X must be Fido. This suggests that the system thinks we have knowledge of all the dogs that exist (the "closed-world assumption"), and if we know X is not any of the $N-1$ dogs, it must be the N^{th} dog. Clearly there may always be a dog we do not know about, and likewise when we learn about a new dog, it does not seem practical to modify the entire set of implications for all the rest of the dogs. What we want is some way of expressing the IS-A relationship between *Dog* and *Fido*. The solution is a connectionist **ontology**.

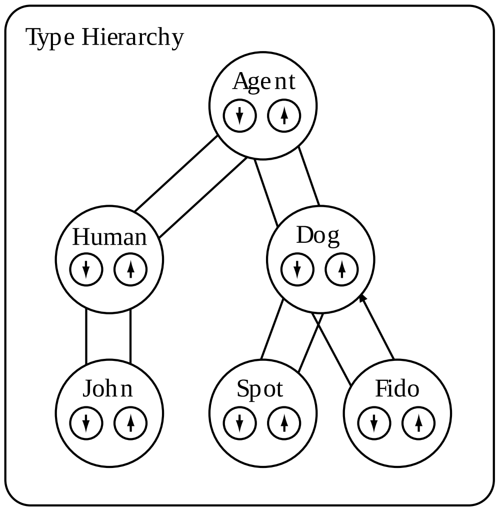

Figure 8: Example of a tiny Ontology

The **ontology** is a hierarchy of types. Each node in the ontology represents a single entity or category and sends activation up to its super-categories and down to its subcategories via two separate paths. No implication rules use nodes in the ontology as their antecedents; instead facts bridge the gap between predicates and entities in the ontology. Although the details will be explained in later sections, let us look at an example of the entire system in practice.

3.3 Detailed example

Let us use an example to see how the system works in some detail. Our model has all the mechanisms described above. Namely, it has a central fluent binder, an ontology (with the entities Agent, Dog, Spot, Fido, Human, and John as shown in Figure 8), and predicates, mediators, and facts containing the knowledge as shown in Figure 9:

- Fact: *Big(Fido)*
- Fact: *CanBite(Dog)*
- *Big(X) & CanBite(X)* $\rightarrow$ *Scary(X)*

To ask the question "Is there a dog that is scary?" we would fill the *agent* role of the *Scary* predicate with a free fluent, say fluent *f1*. We would also use the fluent binder to bind fluent *f1* to the *Dog* cluster in the ontology. Finally, we would activate the *?:Scary* node to indicate a query. This is shown in Figure 10.

Activation of *?:Scary* causes the *?* node of the *Big & CanBite(x)* $\rightarrow$ *Scary(X)* mediator to become active and receive fluent *f1* in its *X* role. This in turn causes the *?* nodes of both the *CanBite* and *Big* predicates to become active with fluent *f1* in their respective roles. This is shown in Figure 11.

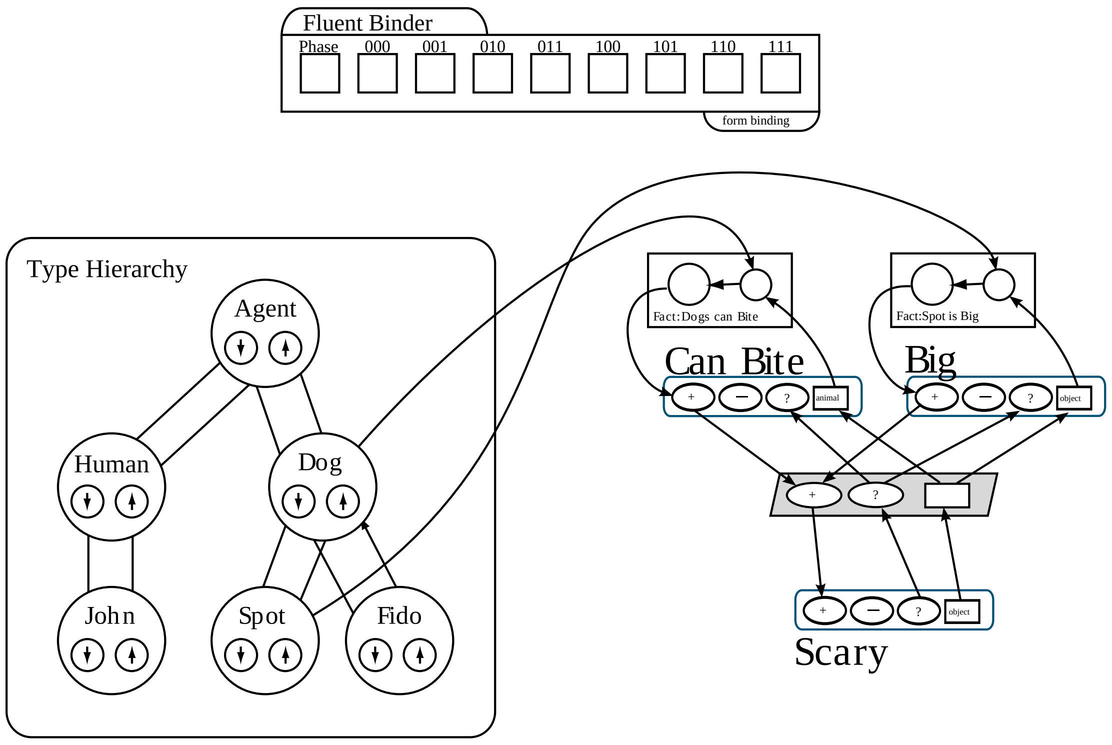

Figure 9: An example network encoding some concepts

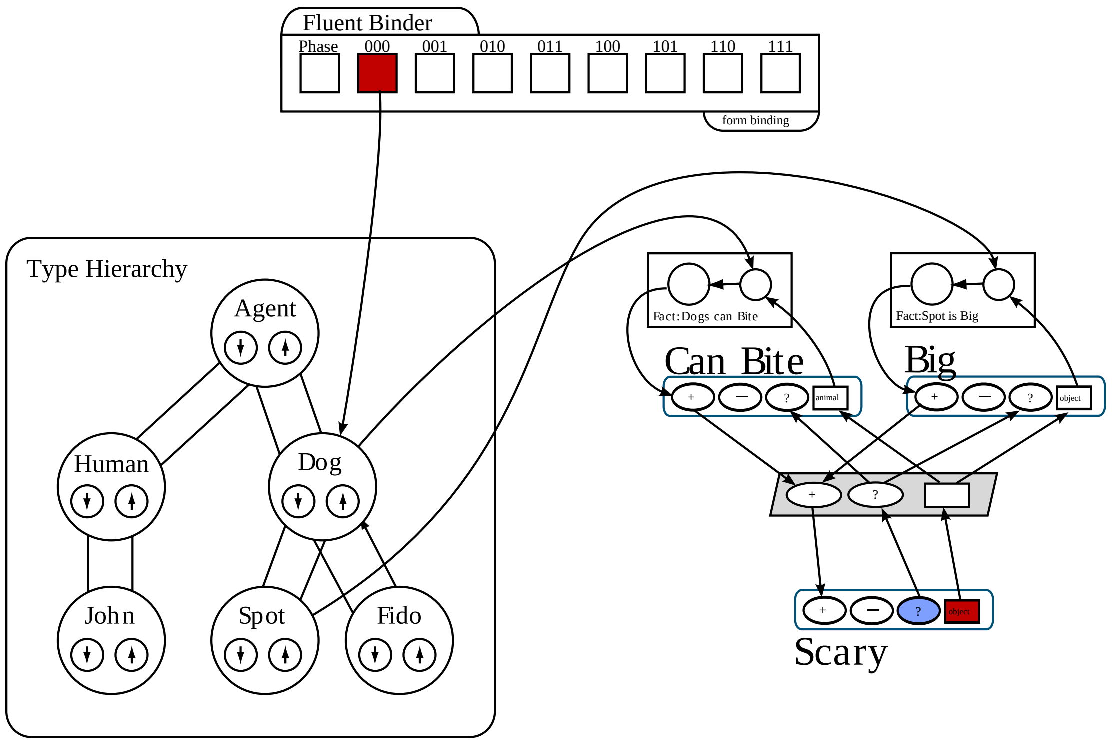

Figure 10: Posing a question to the example network

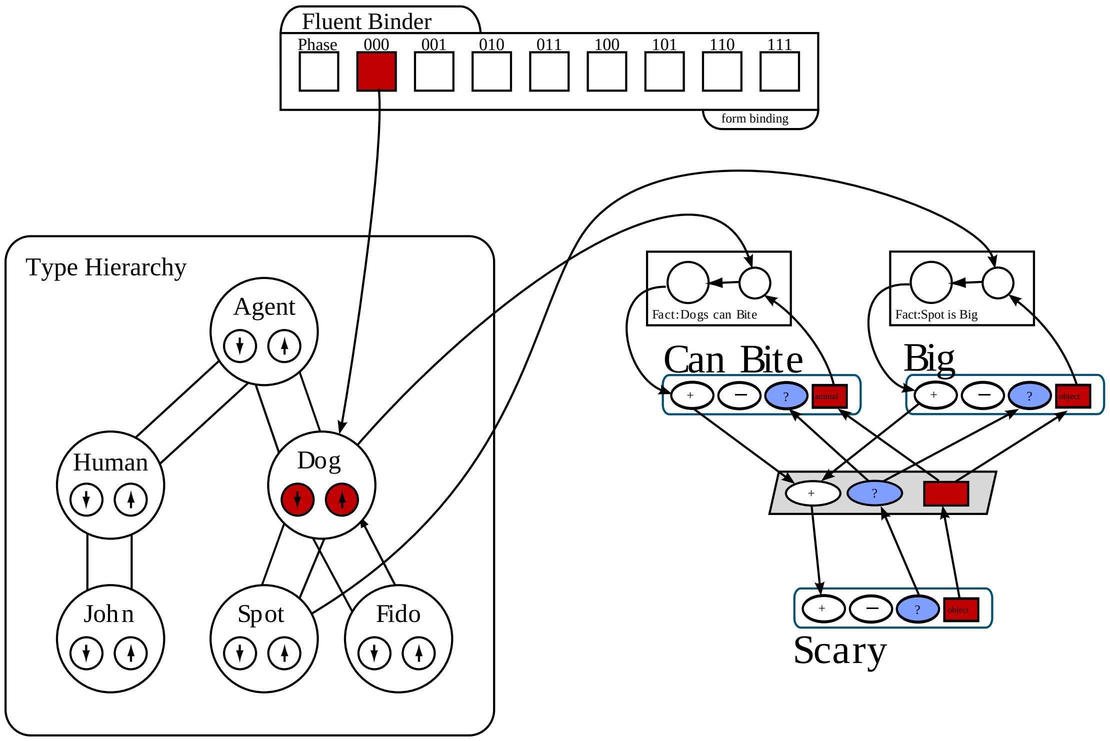

Figure 11: Question activity spreads through the example network

Simultaneously, activation of *f1* in the ontology moves both up and down from *Dog*. Moving up, it activates *Agent*, and moving down, it activates *Spot* and *Fido*. At this point, the fact *Big(Fido)* becomes active because the fluent for *Fido* matches that in the role of *Big*. Therefore, *Big(X)* (which already holds *f1*) has its + node activated. Similarly, the fact *CanBite(Dog)* activates, as may be seen in Figure 12.

Now, the activation of our two facts activates the + nodes of both *Big(X)* and *CanBite(X)*. With both of its antecedents activated, the mediator *Big(X) & CanBite(X)* → *Scary(X)* also gets + activation. Its + node then sends activation to that of the *Scary(X)* predicate. This, then, gives us the answer that there is a scary dog. We then bind *f1* to *Fido*, as it is the only ground entity under *Dog* for which a fact fired. Thus, in Figure 13, the system computes the answer, "Yes, Fido is scary."

3.4 Biological plausibility

Much of the biological plausibility of such a model has already been established by SHRUTI [Shastri, 1999]. However, our additions and modifications must be justified to be biologically plausible. There are two main issues.

First, the central structure that links fluents to represented entities is surprising because we generally observe the brain to be highly parallel, without this sort of centralization. Of course, that observed parallelism is the basis for our connectionist reasoning system. However, experiment has shown that human activity routinely relies on focused attention[Simons and Chabris, 1995], in which a part of the brain, such as the visual system, works as toward a single goal as though its activity is directed as by a central authority. Furthermore, such centralization need not control the entire brain, but only a single region; in our case, only the automatic inference structures would require access to this central mapping structure.

Second, by replacing temporal signals by multi-bit fluent numbers, the structures that pass such signals

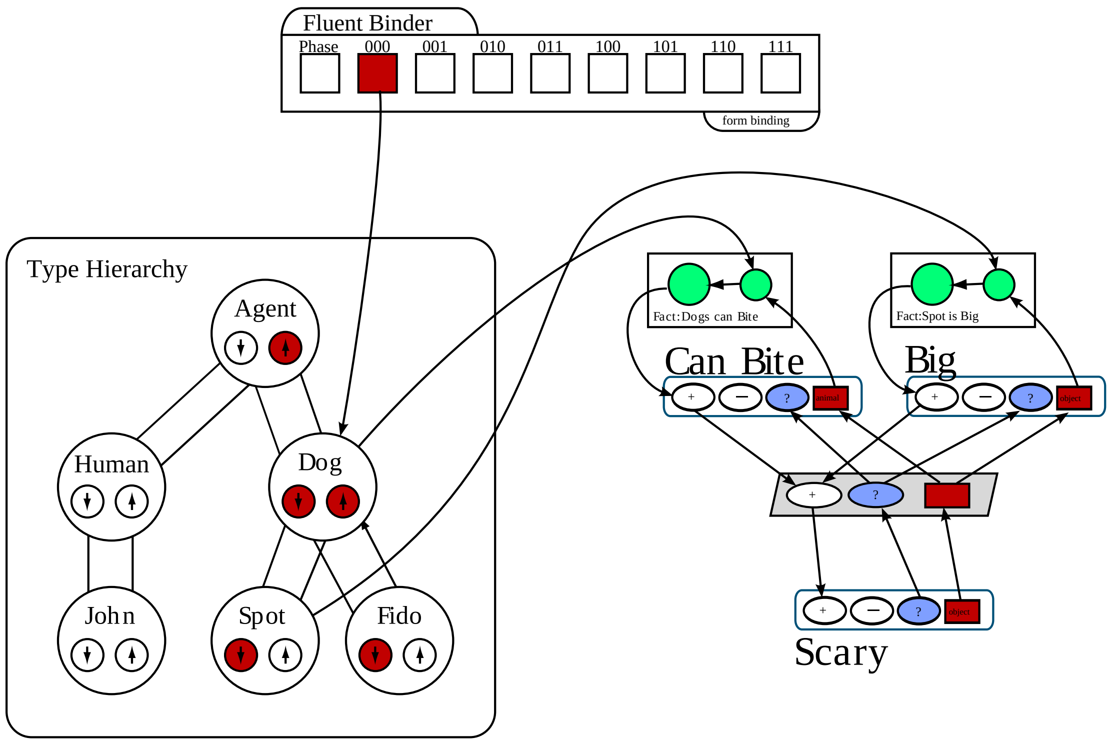

Figure 12: Response activity begins in facts in the example network

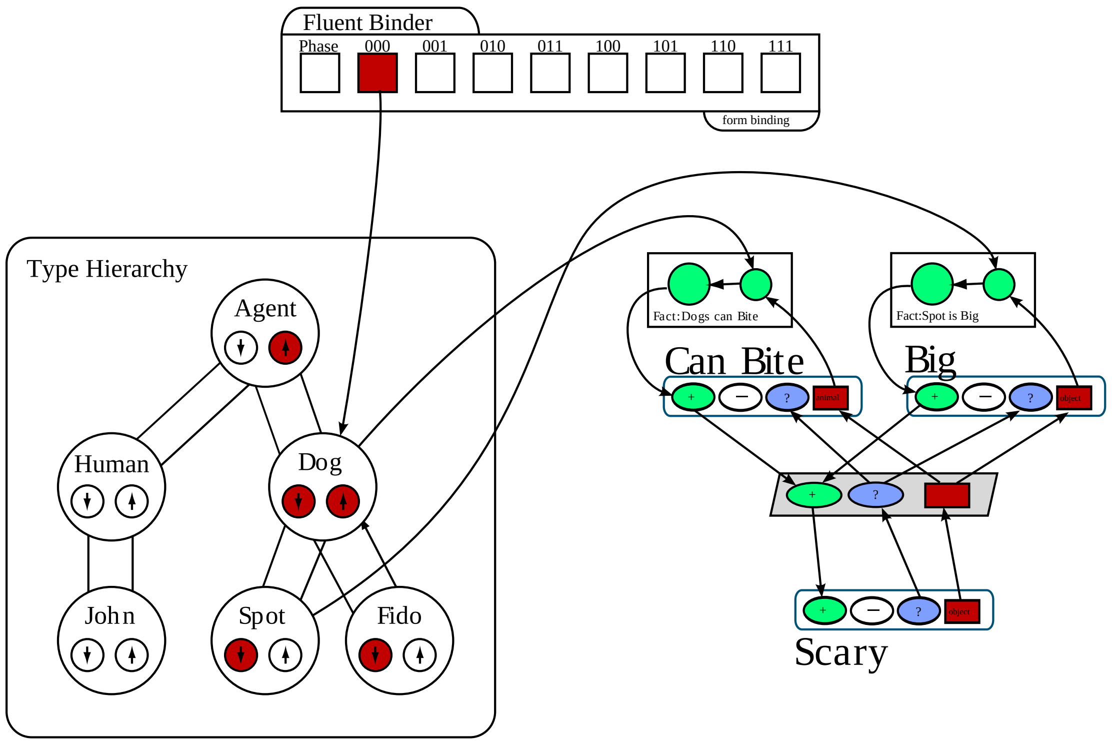

Figure 13: Response activity flows through the network, answering the question

become more complicated. Instead of a single signal being sent to the destination, three or more may need to be sent in parallel, and similar structures must exist in many places through the brain. This is no obstacle; the detailed connections that exist in the visual system, for example, show that biology can lead to detailed neural structures.

4 Computational Level Description of the 6 Tasks of Table 1

Before going further into the details of our implementation in §5, we will show how the connectionist mechanisms described above can be used to carry out the six key tasks described in Table 1.

4.1 Subtask A. Linking a fluent with a fixed item

Linking a fluent to a fixed item or entity is the most basic issue in the fluent method of variable binding. As we saw, long-signature-based approaches to variable binding are implausible because they require passing unique identifiers through the inference process. The fluent idea is to dynamically pass quite small messages (say, 3 bits), which indirectly specify the items involved. However, to provide meaning, these should be attached somehow to representations of concepts or objects. At the computational level, we can imagine a centralized bank of 8 fluent registers, each of which can hold a reference to some item. Everything that follows in this paper relies upon a biologically plausible way of creating a reference to an item.

We suggest two distinct solutions to this problem: The first, A1, exploits standard techniques for dynamic linking, which have been described by others [Kappen and Nijman, 1995]. Essentially, the idea is that short-term potentiation can create a stable path through a network of neurons, providing a link between the fluent and a node in the ontology through which activation can flow. The second solution, A2, uses tree addressing. In A2, the fluent binder's fluent register contains a bit string describing the path through a tree in which the leaves are entities in the ontology. To send activation to that entity, then, activation can be spread through a network that is switched by this tree address, eventually causing only the specified entity to receive it. Neither of these is implemented in the pilot system, but Tsotsos [1990] has built essentially the same tree addressing method as A2.

4.2 Subtask B. Propagate a query though the structured rule and fact set

The Shruti group has worked extensively on connectionist variable binding and inference [Shastri, 1999], and our proposal derives largely from their effort. Like Shruti, we also assume that there are connectionist units that represent inference rules and ones that represent facts about the domain. The rules are represented using "clusters" that contain circuitry for propagating queries and their accompanying variable bindings and also for returning + and - answers and possibly additional bindings. This subtask is to spread activation from a query to all those implications and facts that might help answer it.

The basic design is simple – each predicate cluster represents a logical predicate and has circuitry for receiving both query (*?*) activation and bindings. In our fluent model, each variable position in a cluster is represented by a "role" node that will hold the fluent number currently linked to that variable. The fluent itself may have been assigned a specific item or it might be "open" and need to be filled in as described in §4.4. The crucial idea is to have direct links from each predicate to rules or facts that could verify or refute the logical statement it represents. These links not only spread *?* activation, but also connect variable positions correctly. So, if we had a rule:

$$\textit{Mother(X,Y)} \rightarrow \textit{Child(Y,X)}$$

there would be links from the first position of the *Mother* cluster to the second position of the *Child* cluster and vice versa. Thus, when the *?* node of *Mother* is active, *?Child* will also become active, the first role of *Child* will take on the value of the second role of *Mother*, and the second role of *Child* will take on the first role of *Mother*. Similarly, there are direct links connecting specific facts like *Mother(Hillary,Chelsea)* to the *Mother* predicate cluster; these facts also receive *?* activation from the *Mother* cluster. It is important that specific facts also keep their arguments ordered and keep track of fluent numbers so as to provide correct responses.

With all these mechanisms, the basic algorithm B1 for subtask B becomes fairly simple. First, the constants and variables of the query are assigned to fluents (subtask A). The *?* activation and its accompanying fluent numbers are spread through the connections described above. Each possibly relevant fact will also receive *?* activation and related fluent numbers through these connections. This is process B1.

If there is an answer to the query, the specific facts needed for the answer will be activated at the completion of B1. We will need a more complex variant B2 to include ontologies and this will be described in §4.6.

4.3 Subtask C. Return a + or - answer to a query with all fluents specified

As with subtask B, our solution builds on the Shruti model. Again, each predicate cluster has connections and local state for conveying a true (+) or false (-) answer. We assume that conjunctions of terms are represented as "mediator clusters" as in Shruti. Mediator clusters need only carry a true (+) signal, because the converse could be represented by a second mediator with different weights on its connections. (Later tasks will require additional mechanisms in mediator clusters, however.) If a predicate or mediator cluster then has an answer of any sort, it passes that answer back along the connections going the opposite direction as the question signals.

Unfortunately, there may be recipients of this answer signal that did not send a request for a response; in fact, they may be holding unrelated fluent numbers, representing entirely unrelated information. This could lead to a mixing of signals and interference in the inference system. Fortunately, it is easy to prevent interference. Recall that each cluster was passed a specific fluent number for each of its argument positions as the query activation spread. When return +/- signals are received by a cluster, it compares the accompanying fluent numbers with the ones it has stored. Only if all the fluent numbers match will the cluster send +/- activation further upward towards the original query. A mismatch will lead to this cluster rejecting its current input. Of course, it still might get additional matching responses later from a different inference path, in which case it will send these responses on back to the originating query.

So now we know how responses can propagate. But how do responses get into the system in the first place? A response begins in the fact structures we have described. When a fact receives a query, it gets a both question (*?*) signal and a set of fluent numbers, and it goes into a potentially active state. Next, we want to activate that fact if its fluents' bound entities match the stored entities of the fact. However, it is complicated to have each fact test whether a fluent matches an object, since these bindings are kept centrally. Therefore, we solve this by activating each pair of fluent and bound entity in turn, one at a time, through the central fluent binder. When each pair is active, each fact tests it against its own information. This broadcast-like method allows all fact clusters to check if their bound fluent agrees with their appropriate item; it will be so only if its fluent and entity are active at the same time. If and only if all its fluents and entities match, then the fact sends an answer. Otherwise, it becomes inactive.

4.4 Subtask D. Returning unique answer items as well as + or - answers

We often want to know specific items that satisfy a query, such as *?Color(grass, X)* or *?Gives(Y, Bill, Z)*, which asks who gave what to Bill. Subtask D concerns the restricted case where there is only one answer returned for each open fluent. A moment's reflection should convince you that is inherently difficult for a massive parallel system to work with alternative sets of answers (subtask G), as these sets can grow exponentially. Our proposal on how multiple answer processing might work is described in §4.7.

For this restricted case, the new requirement is that a specific item gets associated with each open fluent (X) in a successful query answer, again assuming for now that only one set of answers is needed. Our solution to this task involves all of the previous algorithms. The first operation is that of Task A—associate a particular item with a specific fluent. The trick is to simultaneously highly activate an open fluent and its filler.

For simplicity, assume that all facts are unary or binary and that each fact is mutually linked to the items involved- specifically this could be through triangle node linkage [Feldman, 1982]. Let us first consider a simple case, the query:

?color(grass, X)

We suppose that grass is linked to fluent *f1*, and fluent *f2* is associated with the open variable *X*. Now algorithm B can be run, yielding activation of all rules and facts that might bear on our query. Assuming that the only relevant fact is:

color(grass, green)

then the node for this fact would be activated and this yields high activation for the item *green*. In this very simple case, the simultaneous activation of *f2* and *green* will suffice to set up the required binding and thus fulfill task D. That is, *green* will be bound to *f2* and *f2* will propagate with a + signal back to the original query. For later use, we assume that an activated fact such as *color(grass, green)*, remains active until it is reset.

Now consider a somewhat harder query *?Gives(W, Bill, Y)* involving two open fluents, say $f3 \sim W$ and $f4 \sim Y$. Suppose that the only relevant fact is: *Gives(Mary, Bill, book7)*.

The system needs to get the right bindings: $f3 \sim Mary$ and $f4 \sim book7$; how can it do this? A key idea, which also be used later, is to activate the system with one open fluent at a time—recall that there are at most 8 of these. Also recall that rules and facts are wired so that argument positions are coherent—the connection between our query and the one relevant fact is structured. Now, the system can probe for the fluent *f3*, corresponding to the *giver* role. The combined activation of the one fact and the first (giver) role leads to high activation of Mary. We initially identified *f3* with the first role, so algorithm A1 could establish the correct binding. The binding of *f4* is similar.

But even this simple case can be tricky. Consider a tiny example with only one rule:

$$Big(X) \text{ and } Can\text{-}Bite(X) \rightarrow Scary(X)$$

and two facts:

Big(Fido)

Can-Bite(Spot)

If the system is given the query *?Scary(X)* it should not, of course, return any positive answer. Our algorithm B1 will activate both of the known facts, but they don't cohere.

For method D1, the system will bind *X* to either Fido or Spot. It isn't important here which is chosen, but let's say it is the one with the most facts involving it currently active. Ties should be broken in some arbitrary way; if ties are broken alphabetically, then the fluent *f1* associated with *X* will be bound to Fido. We would like all facts that are inconsistent with this assignment to become deactivated. This can be achieved by a method similar to B1, which has each potentially relevant fact check to see if all bound fluents agree with its arguments. Here we have just made *f1* no longer open, but bound to Fido. We activate *f1*, which activates Fido as in B1. Any potential fact that does not receive extra activation drops out as before. So, in the end, only one fact is active, and the system correctly refuses to infer that Fido is scary. So, having received a false response, the inference model will repress Fido and search for another binding. This is method D1.

On the other hand, suppose the system had the one additional fact *Big(Spot)*. Then, the two facts involving Spot would cause Spot to be selected for *f1*, and that yields a correct answer that Spot is scary. We will look more carefully at multiple answers in §4.7.

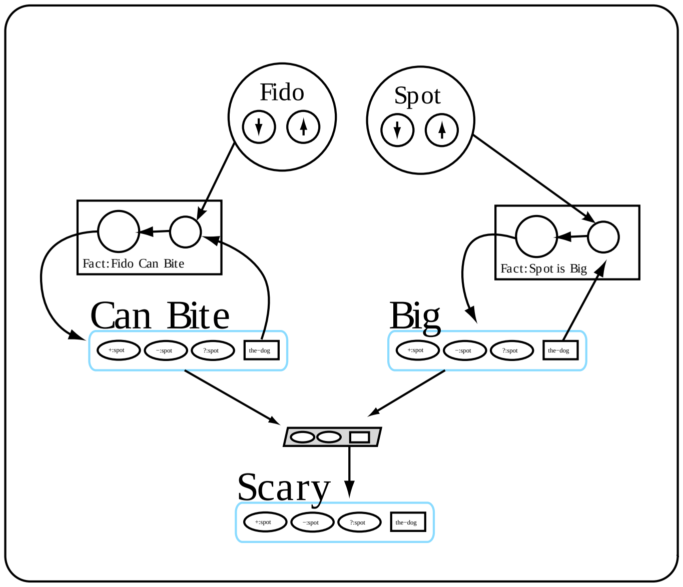

Figure 14: A network demonstrating the issue of task D

4.5 Subtask E. Using multiple copies of clusters

The design so far described employs a cluster and its connections to encode a logical term and its relations to other terms. In a given query, each role of a cluster becomes associated with a fluent. However, it is frequently the case that a query will involve two distinct assignments for the same logical term. For instance, the rule

$$Owns(X,M) \text{ and } Owns(Y,N) \rightarrow CanSwap(X,Y,M,N)$$

requires that two separate instances of the Owns relation, linked to different fluents, are active in the same inference episode. The obvious way to represent this in a network like ours is to have multiple copies of each predicate and mediator.

However, these multiple copies of clusters (predicates and mediators) must be connected together in order for inference to occur, and there are several ways to connect them. Wendelken and Shastri [2004] propose that there be one mediator for each combination of predicate copies in an implication. These are mutually inhibiting, so that only the needed ones are active. Thus, though there is a combinatorial explosion in the number of mediators needed, the types of signals sent are unchanged. For low numbers of multiple copies of predicates the combinatorial increase is perhaps small enough to be plausible.

We explore another option. The key idea is that all copies of a cluster are part of a coherent neural structure that we can call a **family**. The family consists of the individual clusters plus a bit of control logic. At the start of an inference episode, all members of each family are available for binding. When one of these gets *?* activation, the accompanying fluent numbers are assigned to its first copy and this is marked as busy. For another *?* input with the same fluent numbers, nothing changes from algorithm B2, and activation spreads on.

If a family gets a *?* input with fluent numbers different from those stored in the first copy, it assigns them to its second copy. Whenever a family needs more copies than it has, the system fails on that query. We can call this revised query propagating algorithm E1; it extends B1 and is also compatible with adding an ontology as described in the next section.

Interestingly, this family extension requires essentially no change in the answer propagation algorithm C1. This is because clusters already check returning +/- answer inputs to see if the accompanying fluent numbers match the ones it has stored. The extended version just needs each copy to check this. As before, it rejects mismatched fluents and passes on any answers that do match. Details of the current mechanism for multiple clusters can be found in §5.4.

There is a nice efficiency improvement that can be added to multiple cluster models. During inference, activation from a query spreads to all rules and facts that could possibly be involved. Suppose that we introduced a filtering pass between the initial spreading and the return of answers of task D. In particular, we could run process C1, which just returns +/- answers without fluent bindings. Recall that conjunctions are represented by mediator nodes that return activation only if all of their antecedents are satisfied. It could well be that after C1 runs, there are several mediator clusters that fail this test even before particular fluents are chosen. For example, if John is not known to own a car, any conjunction involving John's car will fail.

Now, the system can signal any conjunction cluster that failed under C1 to turn itself off for this episode of reasoning. Then it can again run the initial spreading algorithm, B1 or one of its variants. This could lead to significantly fewer facts being active for step D, which tries to find explicit bindings for each open fluent. This would make the correct binding much faster to find. It could also eliminate some uses of copies of term clusters and thus make it more likely that the system would have enough copies for the query. These efficiency improvements have not been implemented in the current system.

4.6 Subtask F. Adding ontology structure to the set of items

As discussed earlier, it is very common in inference systems to have the items of the knowledge base arranged in a taxonomic hierarchy or lattice, called an ontology. This is intuitively useful because natural similarities and differences can be well-captured by such a type hierarchy. These ontologies can be quite elaborate, but we will look at the simple case where only subtype (ISA) links are represented. So, in our formulation, there might be links from *Fido* to *Dog* to *Mammal* to *Animal* to *Entity.* This new structure adds two kinds of capability to our model system.

Most importantly, an ontology allows us to easily have general facts and rules about classes of objects instead of just rules about specific items. For example, we could have the knowledge *Mortal(Animal)*, and this would also apply to all subtypes of animal. Furthermore, the ontology makes it easy to pose categorical queries, which occur all the time in human reasoning; an example query might be *?Owns(John, Book)*, which asks for a book that John owns, if any.

It is straightforward to extend algorithm B1, which spreads *?* activation to relevant facts, to also exploit the ontology. When the fluent binder activates a node in the ontology, activation then spreads down the hierarchy to lower concepts and specific items and up to more general items. (By using two separate pathways, we can ensure that only sub- and super-types are activated.) This has essentially the same effect as direct activation of the specific items and also activates facts involving intermediate concepts. We call this slightly extended algorithm B2, and this is implemented in the pilot system.

There is some complexity involved in keeping the different fluents separate when activation spreads through the ontology. Each node in the ontology may be activated by any or all of the fluents, and the node must somehow represent which. One solution is to have each node store n copies of its activation information for each of the n possible fluents. Unfortunately, this is exactly equivalent to having n identical ontologies, and is therefore relatively inefficient and hard to learn. Another option is to allow only one fluent access to the ontology at once. This makes it very easy to keep it straight which fluent is activating which entities. However, this adds some complexity to the rest of the system, among other things forcing facts to retain state between phases. We adopt this latter solution for our model system.

4.7 Subtask G. Treating multiple answer sets

All of the discussion above assumes that at most one set of fluent bindings will be returned for a query. One might argue that this is a reasonable model of reflexive memory, but multiple answers are sometimes needed internally for an inference system to find one consistent set of bindings. For intuition, suppose that John and Mary each own n books, and a query asked for a book that they both own. Unless the network is explicitly set up for this type of query, this system would need to test n books to see if there was one that both John and Mary own. In particular, a question that is a conjunction of m predicates may take time exponential in m to answer.

So, we should extend our design to handle multiple answers, either as an intermediate step or to be successively returned as alternative answers to a query. We know that this can not always be done in parallel, but our approximation should be biologically plausible and reasonably efficient. The simplest algorithm that we have found assumes that individual items all can maintain a state of activation over an episode and that this state can be modified by global control.

Given these assumptions, multiple values can be handled. A query will give rise to activation of a number of basic facts, each of which knows which fluent numbers go with each of its arguments, using one the variants of algorithm B. As we saw in Section 4.4, the design needs to choose one item to bind to each open fluent, with the hope of finding a complete match. Our design picks the item that has the highest input from activated facts and that is consistent. After the design has selected one binding for each open fluent, it will try to return a complete answer, following some variant of algorithm C.

If we only wanted a single answer, and one is found, we are done. But if the first attempt fails or if multiple answers are needed, the design must try another set of bindings for the open fluents in the query. There are a number of standard ways to try combinations of variable bindings; we can assume here that the design picks one fluent at random and removes its binding from the competition. This will result in a somewhat different assignment, which can then be propagated back to the query, etc. We call this algorithm G1 and have some ideas on how to make it more efficient. This was not implemented in the current system.

4.8 Unification

In computational realization of logical reasoning or natural language understanding, unification is an important operation. Again, there are various versions of unification, but they all involve identifying two entities that were distinct and then treating them as the same. A simple case might involve recognizing that the Governor of California and the Terminator were the same person. None of the previous connectionist models of variable binding had a way of doing unification, but it is easy to model with fluents.

We assume that each fluent "register" has a way of indicating that has been unified with some other fluent. Unification is then just setting this tag for one fluent and using the other one as usual. That is, the active fluents will get bound to specific entities as part of the answer generation of algorithm D1. Unified fluents will refer indirectly to the entities of their partners. This leads to a minor modification in the algorithm for checking and returning fluent bindings.

In general, when a cluster receives a +/- answer signal, it always needs to compare the accompanying fluent numbers for each position with the ones it has stored from the query. When the unification flag is on, this check also is satisfied if the returning fluent number has been unified with the original one. This modification for task D is called D2. This is another instance where the passing and storing explicit fluent bindings supports functionality not available with time slices. This was not implemented in the current system.

5 Details and Our Implementation

To show the feasibility of a localist inference network like we have described, we implemented a computer simulation with a twofold purpose. Primarily, it is intended to help solidify our model, exposing any weaknesses and causing us to consider every detail. Secondly, the model is intended to be a pedagogical aid to help present the ideas of this paper. In this section, we present some of our insights that were inspired by the creation of a concrete model.

Our simulation is designed to be biological plausible, just like the model. Each model node is simulated by a computational node performing only simple computations. For instance, *+*, -, and *?* nodes just perform weighted sums of their inputs. Similarly, connections between nodes are equally simple, with nodes receiving inputs only from other directly connected nodes. The simulation is turn-based; in each turn, signals are sent and received, and when that is complete, all node states are updated. In particular, node values are transmitted, summed, and processed in a completely localist way.

However, our simulation contains a few non-biological simplifications from the model. An example of this is that, while the model describes the fluent signal as being passed by several simple nodes in parallel (as a string of bits, essentially), in our simulation it is treated as a single signal. Similarly, the complex connections required for properly sending signals correctly in a family of predicates or mediators were not implemented in a truly biological way.

Also, some of the techniques we propose have not been fully implemented in the computer simulation. In particular, the mechanism for binding a fluent to an entity as an answer to a question (e.g. setting *f2∼green* in response to the query *?Color(grass,f2)*) was not implemented. However, this is a technique that is clearly feasible, so it seems unnecessary to fully implement it.

5.1 Fluents and Phases

As described in §4.6, one design difficulty is that the ontology must be used by all fluents without interference, and there are at least two ways to achieve this. Because it seems simple and relatively plausible, we chose to implement a synchronized access method in which only one fluent spreads activation through the ontology at once. This synchronization of the ontology is implemented through the use of **phases**. Each phase represents the activation and ontological implications of a single fluent and its current related entity. Thus, during a phase, the entity corresponding to one particular fluent is activated. Activation spreads from there, representing things related to the fluent's entity. Between phases, signals in the type hierarchy must be inhibited so that the next phase can start without interference. After inhibition, a new phase starts when the fluent binder activates the entity that has been bound to the next phase's fluent. Within the ontology, two separate signals are propagated. One signal only travels up the hierarchy, while the other only travels down. In this way, knowledge about super-categories and sub-categories can be leveraged when activating facts. The fluent binder orchestrates phases through ubiquitous connections throughout the ontology. Note that these phases differ from those used in the temporal synchrony models; here, the phases only modulate activity in the ontology, rather than occurring generally through the inference system.

5.2 Requesting New Fluents

The fluent binder keeps track of the relationship between fluents and entities, but there must also be a biologically plausible way for a new fluent to be requested and associated with a given entity (needed in subtask D, for instance). An example of this happening is when a mediator becomes active without all of the necessary roles being supplied. So, the question *?Own(man, money)* should activate *?Give(agent, man, money)*, where *agent* is a new entity needing a new fluent. The request process occurs in a plausible and scalable way with help from a hierarchical series of winner-take-all networks. At the top level, the fluent binder sends a "fluent-is-free" signal at the start of the phase of any fluent that is unbound. At each level, this signal is passed to the child network with the highest activation. In turn this is passed down until the mediator's role node receives the signal. At this point the mediator (which has a link to its default role filler) activates the appropriate entity in the ontology. This in turn causes the fluent of the current phase to be bound to the given entity. This linking of fluent and entity is subtask A and could be achieved as described above in §4.1.

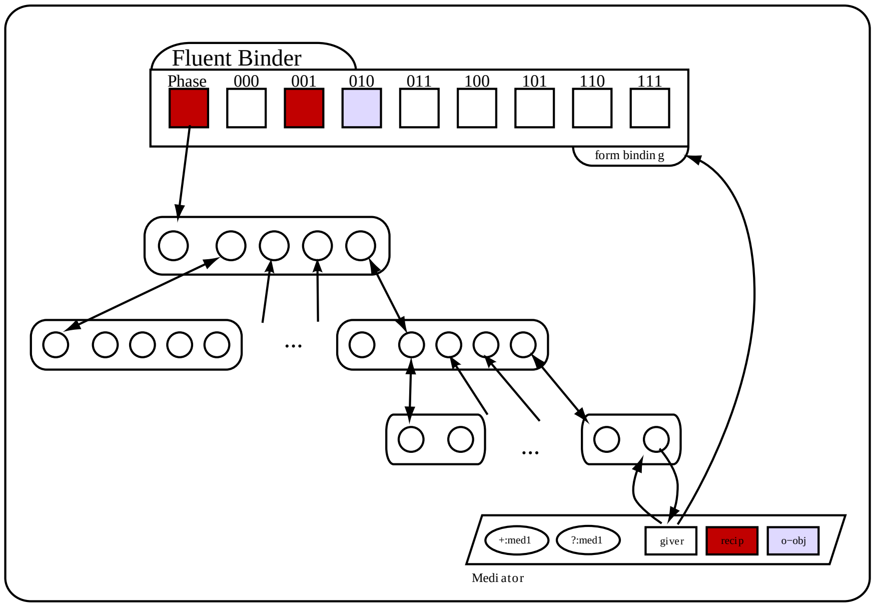

Figure 15: Selection networks allocating free fluents

5.3 Representing Facts

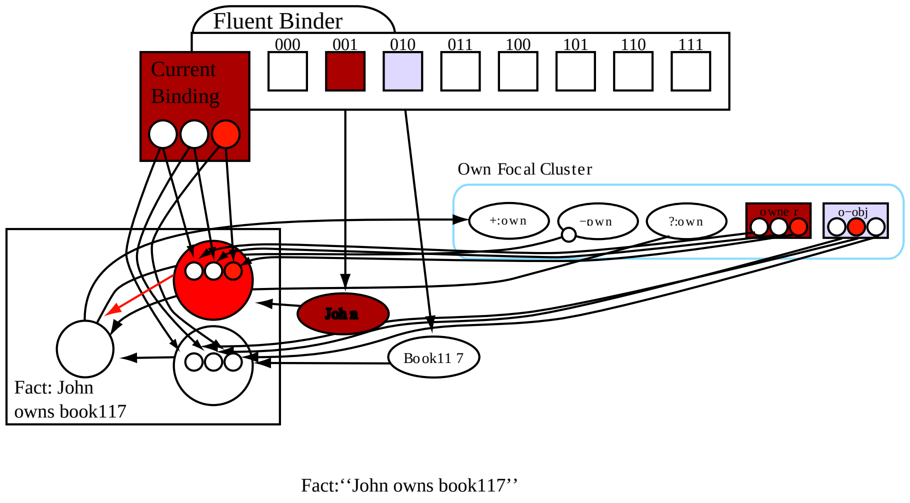

Figure 16: Details of a Fact

The mechanism that facts use to achieve their functionality is straightforward. The circuitry of a fact is a set of constraint nodes, one for each role, followed by a logical AND across those nodes. Activation of a constraint node corresponds to a satisfaction of one the conditions of the fact. In particular, it indicates that particular role in the predicate matches one entity that the fact represents knowledge about. Each constraint node has two incoming links: one from the predicate's role, and the other from an entity in the ontology. If the linked entity in the ontology (in our example, *BillGates*) is activated in the same phase as our role's binding (in our example, *owner*) then the fluent matches the fact's entity, so that role node fires. This firing continues through all of the phases of the fluent binder in order to allow the rest of the constraint nodes to activate as well. Notice that because of the nature of our ontology, super and sub categories of our bound entity will become activated and satisfy the constraint node. In our example, *Man* will become active even if *BillGates* is the entity bound to the fluent in the *Give*-*owner* role, so all relevant facts will be activated.

5.4 Multiple Instantiation

Predicates and mediators must pass information, and this becomes complicated when multiple copies of these clusters are needed. Here, we describe how these messages may be passed—that is, how fluents pass from role to role and computation nodes such as + transmit vales in a coherent way so that these signals are received by the proper cluster. This requires modulating signals to block them from reaching structures where the signals would be useless or confusing.

Some inter-family communications are easier than others. As a trivial case, each mediator family implies exactly one predicate family and, in fact, each mediator corresponds to exactly one predicate. So, it is simple to send responses from the mediator family to the predicate family; one simply connects the response directly from the specific mediator to the corresponding predicate, as we have previously described. The mapping is 1-1, so the messages are easy to route, and the existence of families adds no complications. Similarly, sending questions from predicates to mediators is easy for exactly the same reason.

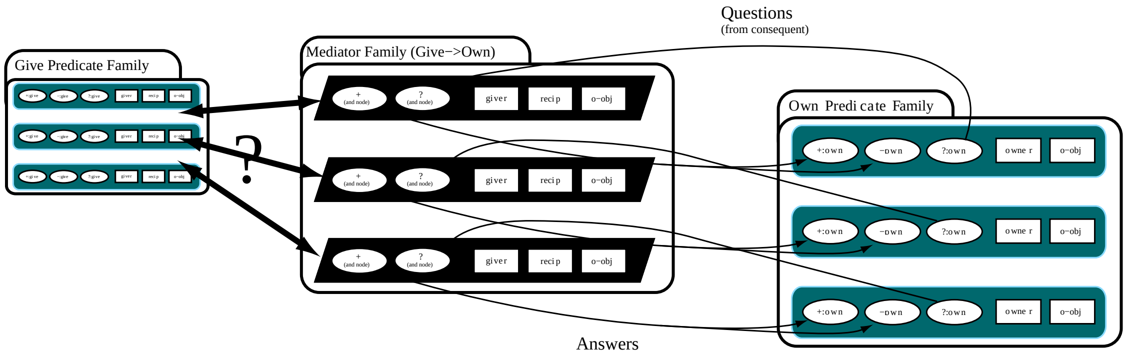

Figure 17: Questions and answers are transmitted 1 to 1 between consequents and mediators because every mediator has exactly one consequent. However the connection between antecedents and mediators is more complicated.

The difficult communications come when mediators ask questions of predicates and when predicates send responses. Each predicate may be queried by many different mediators in many families, and its responses are not necessarily useful to all mediators. These properties make interference likely, so we must be sure that the messages are routed appropriately. This is done with winner-take-all message receivers. We give each predicate family a "question input structure." When a mediator needs to send a question to a predicate, it sends a message to the predicate family's question input structure. The question input structure is a selection network that chooses only a single question input at a time. Then, it routes the question to an unoccupied predicate (such routing can be done easily by a simple network of inhibitory nodes) and continues on to the next question. Thus, it prevents interference between questions to the same predicate family.

Later, when a predicate has received an answer, the predicate sends a message to every mediator in every family that might possibly be able to use the result. All the mediators test for a match, inhibiting this input if any of the fluents of the mediator and predicate do not match (see Figure 19). In this way, responses are

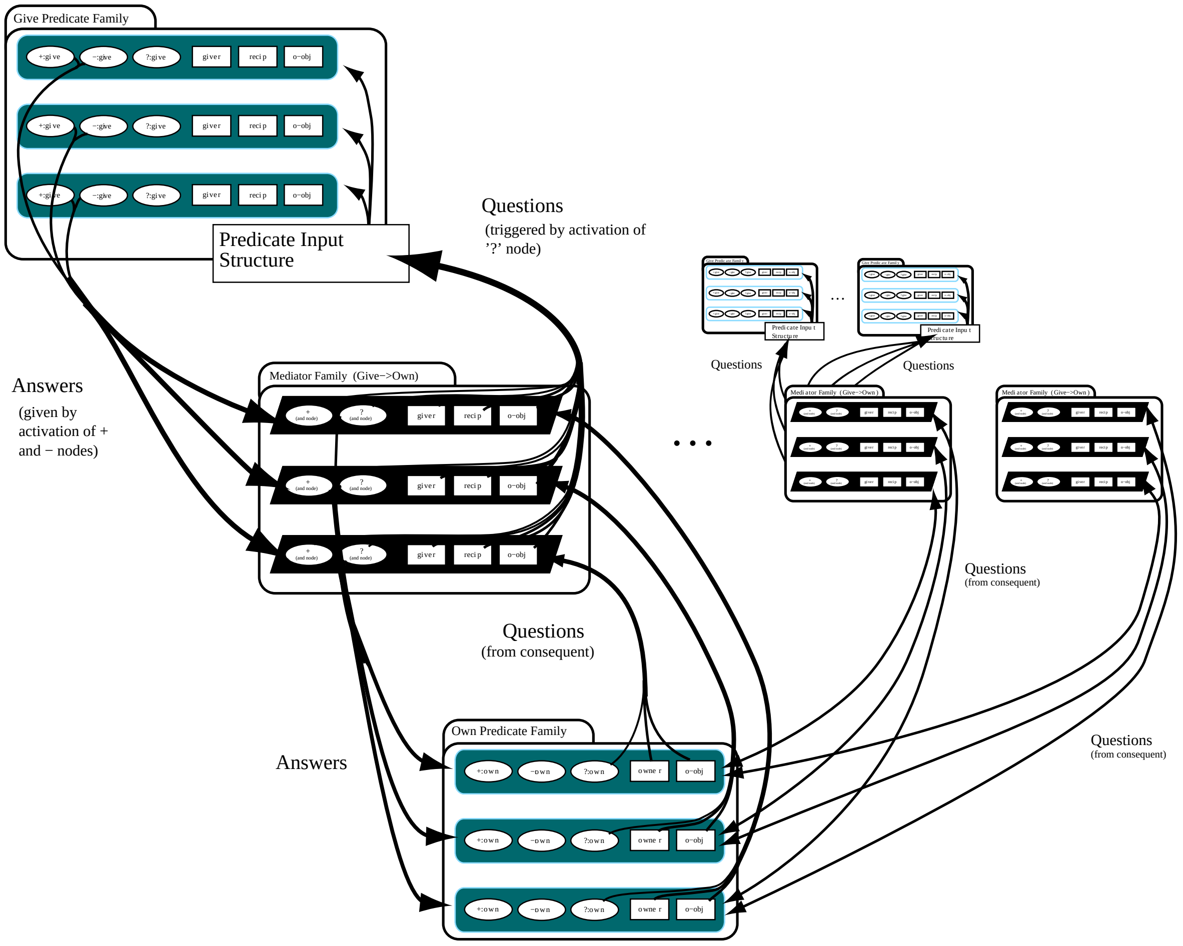

Figure 18: Overview of inference with multiple clusters

routed appropriately. It may seem wasteful to send many answers and then ignore the inappropriate ones, but it is necessary because there is no record kept of which mediator sent the question. (As a benefit, this means that if multiple mediators ask the same question, they all receive the same response.)

5.5 A note on connection weights

At this point the reader may have noticed that there are many links between nodes that have yet to be fully defined. Each edge must have an associated weight corresponding to the ratio of the source node's activation strength to the output node's signal's strength. The weights on many of these edges, such as consequence to mediator and mediator to antecedent can have subtle but profound effects on belief in the network. This can be seen most strongly in the links of facts to their associated predicate. Since there are many facts that may conflict or lead to alternate interpretations (like a Necker cube), appropriate weights on these connections can be critical for accurate and insightful inference. Indeed, much of learning and the "intelligence" of such a system will come from precisely tuned weights on facts and rules. Our current effort is not focusing on connection weights, though this concern is addressed in much other connectionist and other learning theory work.

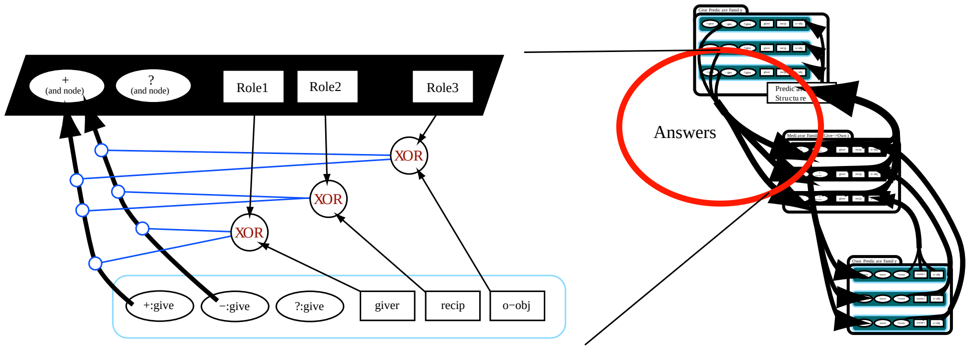

Figure 19: Details of answers

5.6 Automatic inference and interaction with other brain functions

Prior to any input, the system has little activation. The fluent binder is sending free phase signals through the winner-take-all hierarchy, but since no role needs a binding, these signals are ignored. So, to set up a question, these same structures may be used. For an external structure to pose a question, it would bind a fluent, just as a mediator would, and send the question to a predicate, again in the same way as a mediator.

In fact, it seems that one could attach this automatic reasoning system to a language-processing system just by adding simple connections between the language system and our predicates. The language-processing system could even use predicates to store information, and so inferred information might be treated very similarly to information derived directly from language. For simplicity, in our simulation we skip the binding process (as described in §4.1) and automatically set the connection between fluent and entity in the fluent binder.

5.7 Learning

Any serious neural modeling effort should include considerations of the how the proposed structures might arise in the brain. Conceptual knowledge is obviously not innate and thus requires some kind of learning story.

We can assume that a basic architecture of neural clusters like our fluent binder, predicates, and mediators could be part of our genetic endowment. The question becomes one of how they might get wired up correctly as new concepts and relations are learned. The standard structured connectionist approach to this kind of problem is *recruitment learning* [Feldman, 1982, Browne and Sun, 1988]. The basic idea is that there is a pool of uncommitted linking units that can be recruited to build conceptual structure.

The Shruti project has done extensive investigation of learning for their theory. This includes not only computational models and simulations but very detailed analysis of how recruitment learning might map on to the hippocampal complex [Shastri, 2001]. Virtually all of that work can be carried over to the current model. However, there are still a large number of details that would need to be worked out. At this point,

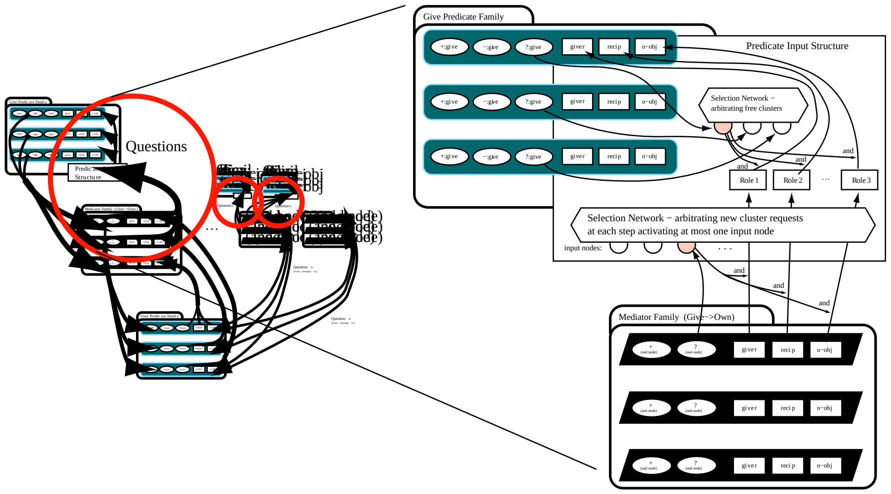
Questions
Give Predicate Family
+:give
~:give
?:give
giver
recip
o−obj
Predicate Input Structure
Selection Network − arbitrating free clusters
and
Role 1
Role 2
Role 3
Selection Network − arbitrating new cluster requests at each step activating at most one input node
input nodes:
Mediator Family (Give−>Own)

Figure 20: Details of the Predicate Input Structure

one can only say that there is no apparent barrier to building a fluent-based conceptual learning model, but it would be at least a doctoral thesis worth of work.

6 Conclusions

It has been clear for decades that variable binding is a difficult task for a massively parallel system. However, human brains need this ability for routine thinking and language use and computational considerations dictate that the mechanism be quite direct. The structures and algorithms in this paper provide a range of suggestions on how our brains might get the effect of variable binding while meeting the biological and computational constraints.

Of course, the model and implemented simulation is only a coarse digital approximation of a complex analog/chemical reality. No one believes that the brain has registers or communicates with binary signal pathways, but the idea of storing and communicating enough information to resolve approximately 8 alternatives is quite plausible. We have suggested a variety of alternate representations and algorithms that are consistent with known results. It would be great to get experimental data that help choose among alternatives, but even before that more computational work can help provide further insight into the brain.

On a final note, we observe that our model, like Shruti and others, is computationally rather complex. Before dismissing such complexity as implausible, however, note the high complexity of other biological processes, such as blood clotting and neural development. Clearly, the human body behaves in many complex ways, and we should not be appalled that our inference processes may be similarly complicated.

References

D. H. Ballard. Parallel logical inference and energy minimisation. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 203–208, 1986.

A. Browne and R. Sun. Connectionist recruitment learning. *European Conference on Artificial Intelligence*, pages 351–356, 1988.

A. Browne and R. Sun. *Connectionist variable binding.* Springer Verlag, Heidelberg, 2000.

A. Browne and R. Sun. Connectionist inference models. *Neural Networks*, 14:1331–1355, 2001.

Jerome Feldman. Dynamic connections in neural networks. *Biological Cybernetics*, 46:27–39, 1982.

Ray Jackendoff. *Foundations of Language.* Oxford University Press, Oxford, 2002.

H. Kappen and M. Nijman. Dynamic linking in stochastic networks. In Moreno-Diaz R., editor, *Proceedings W.S. McCullock: 25 years in memoriam*, pages 294–299, Las Palmas de Gran Canaria, Spain, 1995. MIT Press.

T. Lange and M. Dyer. Frame selection in a connectionist model. In *Proc. 11th Cognitive Science Conf*, page 706 713, Hillsdale, NJ, 1989. Lawrence Erlbaum Associates.

P.M.V. Lima. *Logical abduction and prediction of unit clauses in symmetric hopfield networks.* Elsevier, Amsterdam, The Netherlands, 1992.

P. R. V. Looke. Qnet: A quantum mechanical neural network. *Cybernetica*, 38(1):85–106, 1995.

R. O'Reilly, R. Busby, and R. Soto. Three forms of binding and their neural substrates: Alternatives to temporal synchrony. Technical report, Department of Psychology University of Colorado at Boulder, 2001.

Lokendra Shastri. Advances in Shruti — a neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. *Applied Intelligence*, 11:79–108, 1999.

Lokendra Shastri. A computational model of episodic memory formation in the hippocampal system. *Neurocomputing*, 38:889–897, 2001.

D. Simons and C. Chabris. Gorillas in our midst: Sustained inattentional blindness for dynamic events. *British Journal of Developmental Psychology*, 13:113–142, 1995.

R. Sun. A discrete neural network model for conceptual representation and reasoning. In *Proceedings of the 11th Conference of the Cognitive Science Society*, pages 916–923, Hillsdale, NJ, 1989. Lawrence Erlbaum.

R. Sun. On variable binding in connectionist networks. *Connection Science*, 4:93–124, 1992.

J. Tsotsos. Analyzing vision at the complexity level. *Behavioral and Brain Sciences*, 13(3):423–445, 1990.

C. Wendelken and L. Shastri. Probabilistic inference and learning in a connectionist causal network. In *Proceedings of the Second International Symposium on Neural Computation Berlin*, Germany, 2000.

C. Wendelken and L. Shastri. Multiple instantiation and rule mediation in SHRUTI. *Connection Science*, 16:211–217, 2004.