Beyond Associative Memories: Logics and Variables in Connectionist Models

Ron Sun
Honeywell SSDC
3660 Technology Drive
Minneapolis, MN 55418

January 29, 1992

Abstract

This paper demonstrates the role of connectionist (neural network) models in reasoning beyond that of an associative memory. First we show that there is a connection between propositional logics and the weighted-sum computation customarily used in connectionist models. Specifically, the weighted-sum computation can handle Horn clause logic and Shoham's logic as special cases. Secondly, we show how variables can be incorporated into connectionist models to enhance their representational power. We devise solutions to the connectionist variable binding problem to enable connectioninst networks to handle variables and dynamic bindings in reasoning. A new model, the *Discrete Neuron* formalism, is employed for dealing with the variable binding problem, which is an extension of the weighted-sum models. Formal definitions are presented, and examples are analyzed in details.

ACKNOWLEDGEMENTS I wish to thank Dave Waltz, James Pustejovsky, and Tim Hickey (all of Brandeis University) for many discussions. I also want to acknowledge Honeywell SSDC for the support during the revision of this paper. Finally I wish to thank the reviewer and the editor for their detailed comments.

1 Introduction

Neural networks (connectionist models) have demonstrated capabilities as associative memories and for performing arbitrary mappings. However, as many researchers pointed out, an associative retrieval or a mapping function does not constitute real intellegence in any significant sense, notwithstanding its power and universality. Work done under connectionism is largely geared towards demonstrating that these simple network models have capabilities beyond those of retrieval or mapping, despite the fact that the two are most widely explored and are most directly applicable in application domains. A number of topics have been explored, under the banner of connectionism, to extend the capability of neural networks, including natural language processing, planning, control, and logical reasoning. These efforts together amount to showing that, while the associative memory and functional mapping are the most significant aspects of neural network/connectionist models, connectionist models have more to offer in modeling intelligence and in building artificially intelligent machines.

This paper is concerned with one important part of this endeavor, namely, logical reasoning, which is a significant aspect of human intelligence (Collins & Michalski [6], Nilsson [16], Hayes [13]) and presumably also constitutes a significant aspect of machine intelligence. There is a long tradition of psychological study of reasoning using logical approaches. For example, Jean Piaget attempted to build models of human reasoning in which logic serves as a foundation. In his highly influential theory, the cognitive development in children is described in terms of acquiring a system of mental logic gradually, and changes are interpreted as emergence of new logical competence (Girotto & Legrenzi [12]). In Pinker & Prince [17], the lack of rules (presumably in some kind of logical forms) and variable binding is considered to be a major shortcoming, or even a serious handicap, for connectionist models. They believe that rules, modular structures and variables are essential for human language faculty, and are thus important to be taken into account in connectionist models of such processes. Although formal logic is overrestrictive to account for all kinds of reasoning [1] and has only limited success

[1]Formal logic is too restrictive to account for human reasoning, because it has to have all

to date toward the grandiose goal of accounting for all intelligence, real or artificial, it is still an important and basic component of intelligence to be reckoned with. In artificial Intelligence research, *production systems* (a form of logical reasoning) are the predominent form (e.g. Buchanan & Shortliffe [5]).

Formal logics are simple, formally defined languages, capable of expressing rules in a rigorous way, the essential purpose of which is to formalize in symbolic terms the reasoning processes in mathematics and in daily life. There are basically two types of formal logics: propositional logic and predicate logic. Propositional logic deals with declarative sentences (propositions) that can be either true or false. Propositions can be combined, using logical connectives, to form compound propositions. Many different axiomatic systems have been studied regarding the truth value of a proposition or the relations between truth values of different propositions. Rules, which is an important component of reasoning, are expressed as logical implications; that is, they have to follow the formal definition and axioms regarding logical implications. Usually rules are encoded using the Horn clause formalism, in which conditions are the conjunction of atomic propositions, and conclusions are single atomic propositions. For example, $a \cap b \longrightarrow c$. Predicate logic (first order logic) is made up of predicates which contain arguments, terms for arguments and quantifiers constraining arguments. This type of logic is more complicated because of the introduction of arguments and consequently its binding to terms, but it is more expressive, and it can represent n-ary relations. Formal logics can serve as yardsticks for measuring the logical reasoning capability of connectionist models.

When discussing connectionist models in relation to logical reasoning (or just logic like reasoning), often issues related to variables and variable binding are omitted, in order to simplify the discussion. However, variables are essential for connectionist systems, as indicated by many leading researchers (Feldman & Ballard [9], Pinker & Prince [17], and Fodor & Pylyshyn [10]). And the problem of incorporating variable binding into connectionist models is considered to be a difficult problem. As in Feldman & Ballard [9] the following example demonstrates the necessity of having variables: suppose we have a red circle and a black square,

conditions precisely specified in order to perform one step of inference, and it does not take into account the gradedness of concepts, propositions, and rules, etc.

we have to have a way to associate red with circle and black with square; this binding has to occur and cease dynamically when a system is reasoning with (visual) input data. Variable binding is essential in creating dynamic structures in connectionist networks. Early research in connectionist symbolic processing and high level connectionist models avoids the variable binding problem by reasoning with one object at a time or by using multiple instantiations, such as in Gallant [11], Shastri [18], Kosko [14], Derthick [7], and Blelloch [4]. This approach limits the processing power of the respective systems, and increases greatly the computational complexity. So, in order to enhance the representational and reasoning capability of connectionist models, we need to have variables, and thus we need variable binding mechanisms. It is important to solve the variable binding problem for achieving greater computational power in connectionist systems.

A few attempts have been made to incorporate logical reasoning into connectionist networks and to solve the variable binding problem (see [25], [26]). Touretzky & Hinton [29] is the first piece of research that directly attacks the problem. Their system is made up of several modules or subnetworks. With distributed representation in these modules, they are able to match variables against data in the working memory by *winner-take-all*. The system has very complex structures and is costly in terms of computational resources, but it uses only very simple node functions. It serves as an "existence proof" of solutions to problems of introducing rules, variable bindings and other symbol processing capabilities into connectionist models. Another piece of early work is Smolensky [21], which proposes using tensor products for variable binding, with either local or distributed representation. Basically, each combination of variables and values is represented in the system by a single processing element, or a group of them, which detects dynamically if a binding is present. It utilizes simple node functions but highly complex networks.

Ajjanagadde & Shastri ([1] and [19]) present systems that perform backward and forward chaining inference with variable binding handled by temporal coding of two different types. The inference is made by propagating signals along links in the network. The variable binding mechanism is more complex. The later version of the binding mechanism basically works this way: a node representing a variable produces a temporal activation pattern with a specific frequency which represents a particular binding, when inputs are received that indicate the node should produce that binding. This scheme has relatively simple network structures but needs complex node functions (with temporal properties). Observing the fact that only a temporal frequency is used for representing a particular binding and there is nothing else to it, it follows that this scheme is isomorphic to the system in Sun [22] and in Sun & Waltz [28] by mapping a particular frequency into a numerical value to be passed along (to be described later). [2] Lange & Dyer [15] present another scheme for making inferences in connectionist networks similar to this. Other related work includes Dolan & Smolensky [8] and Barnden [3].

The plan for the rest of the paper is as follows: In order to understand better the relationship between logics and connectionist models, we look into a logic FEL, which directly maps onto the weighted-sum connectionist models and links connectionist models to logics. Then we work out a solution to the variable binding problem, presenting a formal model of node activations and developing mechanisms for dealing with some difficult issues in variable binding.

2 Logics and Connectionist Models

We will look into the question of how connectionist models (with or without variable binding) are related to logics, in order to understand the scopes and correctness of connectionist implementations of logical reasoning. We will look into a "propositional" logic FEL, which stands for *Fuzzy Evidential Logic* (cf. Zadeh [30]), and its relation to weighted-sum connectionist models and (propositional) Horn clause logic. FEL is in fact a direct reformulation of the weighted-sum connectionist models in logical terms (Sun [24]), and thus whatever can be handled by FEL can be dealt with by connectionist models. Here are some formal definitions:

Definition 1 *A* **Fact** *is an atom or its negation, represented by a letter (with or without a negation symbol) and having a value between l and u. The value of an atom is related to the value of its negation by a specific method, so that knowing*

[2]With such numerical values, this solution therefore requires little extra processing in each node (without the need for temporal properties), while at the same maintains a simple network structure.

the value of an atom results in immediately knowing the value of its negation, and vice versa. [3]

For example, a, ¬a, x, and m are all facts.

Now we can define rules and their related weighting schemes:

Definition 2 *A* **Rule** *is a structure composed of two parts: a left-hand side (LHS), which consists of one or more facts, and a right-hand side (RHS), which consists of one fact. When facts in LHS get assigned values, the fact in RHS can be assigned a value according to a weighting scheme* [4]*.*

For example, $a\ b\ c \longrightarrow d$ is a rule, and a weighting scheme associated with it will tell us the value of d once we know the values of a, b and c.

Definition 3 *A* **Weighting Scheme** *is a way of assigning a weight to each fact in LHS of a rule, with the sum of the absolute values of all weights less than or equal to 1, and of determining the value of the fact in RHS of a rule by a thresholded (if thresholds are used) weighted-sum of the values of the facts in LHS (or inner-products of weight vectors and vectors of values of LHS facts). When the range of values is continuous, then the weighted-sum is passed on if its absolute value is greater than the threshold, or 0 if otherwise. When the range of values is binary (or bipolar), then the result will be one or the other depending on whether the weighted-sum (or the absolute value of it) is greater than the threshold or not (usually the result will be 1 if the weighted-sum is greater than the threshold, 0 or -1 if otherwise).* [5]

For example, a weighting scheme for the above rule is

$$w_1 = 0.3, \quad w_2 = 0.3, \quad w_3 = 0.4.$$

and the value of d, which is between l and u (e.g.$[-1,1]$ or $[0,1]$, depending on the value ranges of a, b, and c), is calculated by their weighted-sum (or inner-product):

$$d = w_1 a + w_2 b + w_3 c$$

[3]We will adopt a generic confidence measure as the value of a fact. We will later further characterize this measure.

[4]When the value of a fact in LHS is unknown, assign a zero as its value.

[5]This weighting scheme can be generalized, as will be discussed later on.

Definition 4 *A* **Conclusion** *in FEL is a value associated with a fact, calculated from rules and facts by doing the following:*

(1) for each rule having that conclusion in its RHS, obtain conclusions of all facts in its LHS (if any fact is unobtainable, assume it to be zero); and then calculate the value of the conclusion in question using the weighting scheme;

(2) take the MAX of all these values associated with that conclusion calculated from different rules or given in initial input.

Note that *conclusion* is defined here recursively. Thus, depending on the structure of a particular theory, this definition may or may not make sense (e.g., in case of recursive structures in a theory). In order to use this definition and the procedure for calculating conclusions implied by the definition, we can restrict the structure of theories allowed.

Definition 5 *A rule set is said to be* **Hierarchical**, *if the graph depicting the rule set is acyclic; the graph is constructed by drawing a unidirectional link from each fact (atom) in LHS of a rule to the fact (atom) in RHS of a rule.*

Making a rule set hierarchical avoids circular reasoning.

Now FEL can be defined as follows:

Definition 6 *A* **Fuzzy Evidential Logic** *(FEL) is a 6-tuple: $< A, R, W, T, I, C >$, where A is a set of facts (the values of which are assumed to be zero initially), R is a set of rules, W is a weighting scheme for R, T is a set of thresholds each of which is for one rule, I is a set of elements of the form (f, v) (where f is a fact, and v is a value associated with f), and C is a procedure for deriving conclusions (i.e., computing values of facts in RHS of a rule in R, based on the initial condition I).*

We want to show here the generality and the power of FEL. We will first show that FEL can simulate Horn clause logic.

Definition 7 *A* **Binary FEL** *is a reduced version of FEL, in which values associated with facts are binary, the sum of weights (each of which is positive) in each rule is 1, and all thresholds are set to 1.*

Here is the theorem for the equivalence (see [24] for proof):

Theorem 1 *The binary FEL is sound and complete with respect to Horn clause logic.*

Having dealt with Horn clause logic, we want to show that FEL can handle as well an extension of Horn clause logic – Shoham's logic (Shoham [20]; a modal logic formalism purporting to explain causality), to further demonstrate the power of FEL. We will first define the Causal Theory (CT) (cf. [20]):

Definition 8 *A* **Causal Theory** *is a set of formulas of the following form*

$$\wedge_i \Box n_i a_i \wedge_j \Diamond n_j b_j \longrightarrow \Box c$$

where n_i's are either $\neg$ or nothing, $n_i a_i$'s are necessary conditions (causes), and $n_j b_i$'s are possible conditions (enabling conditions). C is concluded iff all $n_i a_i$'s are true and none of $n_j b_j$'s are known to be false.

This is a non-temporal version of CT. Informally, causes are primary conditions which, together with other conditions, will bring about the effect. These "other" conditions are somewhat secondary. In reasoning, as long as we know that the primary conditions (*causes* or *necessary conditions*, preceded with $\Box$) are true and that there is no information that the secondary conditions (*enabling conditions* or *possible conditions*, preceded with $\Diamond$) are false, then we can deduce that effects will follow. This definition makes the theory non-monotonic, because if later on one of these secondary conditions become known to be false, then the reasoning has to be retracted somehow.

Theorem 2 *For every hierarchical, non-temporal Causal Theory, there is a FEL such that CT: $w \models a$ iff FEL: $c \models$ 'a=1', where w is a set of initial conditions for Causal Theory CT, and c is the set of initial conditions for FEL mapped over from w in CT; $\models$ denotes derivability as usually defined.*

For proof and other details, see [24].

The advantage of FEL lies in the fact that, in spite of its generality, it can be implemented in connectionist networks directly, and thus it bridges the gap between logics and connectionist models. To link FEL to connectionist models we define an "implementation" of FEL:

Definition 9 *An* **Implementation** *of FEL is a network of elements connected via links, where each element represents an atom and its negation (there is a one-to-one mapping between an atom and an element) and links represent rules, going* *from elements representing facts in LHS of a rule to elements representing facts in RHS of a rule.*

We define the notion of an *element* formally:

Definition 10 *An* **Element** *is a structure that represents one and only one fact and has multiple sites each of which receives a group of links that represents one single rule.*

This implementation of FEL is in fact a connectionist network: elements are nodes in a connectionist network, rules are, as defined above, links between these nodes emenating from nodes representing conditions in a rule to nodes representing conclusions in a rule; and weighted-sum computation is carried out within each node for evidential combination in reaching conclusions based on the values of conditions. Through FEL, the logical capability of the weighted-sum connectionist models is established.

FEL, like any other propositional logic, has arguments hidden in propositions. For example, "Human is mortal" can be expressed in propositional FEL as

$$human \longrightarrow mortal$$

This can be easily extended into a predicate version:

$$human(x) \longrightarrow mortal(x)$$

which means that if x, which is an argument to be bound to something, is *human*, then x is mortal. Other rules and facts we presented before can all be turned into corresponding predicate versions in exactly the same way.

Besides this simple single argument case, we can also deal with multiple arguments in one predicate in a similar way. For example, to express "if one locale is north to another, then that locale may be colder than the other." we use the following predicate rule:

$$north\text{-}to(x, y) \longrightarrow colder\text{-}than(x, y)$$

Of course, each of these predicate rules should have weights associated with them when expressed in FEL.

Let us describe formally the predicate version of FEL:

Definition 11 *A* **Fact** *is an atom or its negation, represented by a letter (with or without a negation symbol), with a fixed ordered set of arguments associated with it and having a value between l and u.*

Definition 12 *A* **Rule** *is a structure composed of two parts: a left-hand-side (LHS), which consists of one or more facts, and a right-hand-side (RHS), which consists of one fact. When facts in LHS get assigned values, the fact in RHS can be assigned a value according to a weighting scheme. Any binding associated with a argument in LHS will be assigned to the arguments with the same name in RHS.*

Definition 13 *A* **Conclusion** *in FEL is a value associated with a fact, calculated from rules and facts by doing the following:*

(1) for each rule having that conclusion in its RHS, obtain conclusions of all facts in its LHS, or assume some values to be zero if they are unobtainable; and then calculate the value of the conclusion according the weighting scheme;

(2) take the MAX of all these values associated with that conclusion calculated from different rules or given in initial input.

(3) for arguments in the fact, assign correct bindings, if they exist, from the corresponding arguments in the LHS of the winning rule (the one having the MAX value).

FEL can be defined similarly based on the new definitions of rules and facts, etc.

In FEL, unlike in Prolog or other logic programming languages, we prefer not to allow functions to be bound to arguments (it can be done nevertheless; cf. Sun [24]). In other words, there is no function in our universe of discourse. Also, we in principle do not handle recursions; the reason is that we are aiming at modeling commonsense reasoning, not logic programming, so the loss in computational power is more than compensated by the resulting efficiency in reasoning.

In short, FEL helps to clarify the logical capability of connectionist models, with or without arguments/variable binding, in a very direct way (instead of going through Turing computability). So we know for sure that weighted-sum connectionist models are at least as powerful as Horn clause logic or Shoham's logic, and much more (e.g. it can deal with graded concepts and uncertain rules; cf. Sun [24]).

3 Variable Binding

Now the question is: how do we incorporate arguments (for predicates) into connectionist networks; in other words, how do we deal with the variable binding problem? We will first present informally the solution to the variable binding problem and a formal treatment based on the DN (the *Discrete Neuron*) model is then developed. A detailed example will be analyzed.

Why is it difficult to introduce variables into connectionist models? Recall that in conventional connectionist models, each node performs very simple processing, usually just weighted-sum or similar numerical calculations, and structurally these models are uniform and homogeneous. Therefore, they are inherently structureless and it is extremely difficult to impose any symbolic structures onto them because of the uniformity and homogeneity. The problem of introducing variables into connectionist models amounts to the questions of how to add structure to networks without losing too much of that characteristic uniformity and homogeneity and how to enhance the processing power of each processing element to a degree sufficient for symbolic processing while maintaining its simplicity. In other words, our aim here is at devising a network structure as simple as possible while maintaining well-definedness and relative simplicity of node functions at the same time.

For now, without variable binding, we have the following picture: if there is a rule $a\ b\ c \longrightarrow d$, then in the network that incorporates the rule, there is a link from each of the conditions (a, b and c) to d, and these three links together represent a rule. Each node performs a weighted-sum computation of its inputs from incoming links, for adding up its evidential support (confidence values). Once activated, a node will pass on its own activation (i.e., confidence value) to nodes downstream, i.e., nodes which this node is connected to as a condition in a certain rule (and consequently these nodes are conclusions of the rules involved; see Sun [24]). Justifications for this interpretation of weighted-sum connectionist models have been given before, in terms of relationships between logic and connectionist models. We will now add variables into the above connectionist network.

3.1 Representing Variables

To add variables, first of all we have to address the question of how to represent variables in connectionist networks. Each variable in each fact (predicate) can be handled by one node. The variable can become bound by an application of a relevant rule that has in its RHS the fact that contains the variable. The binding is done by a numerical value, which represents a particular constant (a binding), being passed along the link (which represents the rule in question) to the node representing the variables, from a node representing a variable having the same name in a fact in the LHS of the rule applied. For example, suppose we have a rule: "if an area grows something, it is an agricultural area." It can be expressed as follows:

$$grow(x, y) \longrightarrow agr(x)$$

and the input is grow(a, b) (meaning *area a* grows *b*), where a and b are constants, e.g., a=Florida and b=orange. The binding a is passed to the node representing x, and b to the one representing y in grow(x,y). Then by applying the rule, a is further passed [6] from the node representing x in grow(x, y) to the node representing x in agr(x).

Let us see how this can be done. The same network structure can be kept just as before, but in place of a node now we have an *assembly*, or a collection of nodes interconnected in some way. An assembly should consist of a certain number of nodes, one of which is for computing and storing values of predicates (we will call it the C node), and the rest are for variables (we will call them X nodes). Suppose we have k variables, an assembly then contains k+1 nodes, each of which except one is for storing and passing along bindings of the variable which it represents. Among these nodes, the C node of an assembly receives inputs from C nodes in other assemblies which are conditions (LHS) of a rule that use this assembly for its RHS. Each variable (X) node receives inputs from variable (X) nodes in the same set of assemblies as its corresponding C node, picks up its own binding and passes

[6] However, in order to do this, we need to extend the idea of node activation functions to allow the passing and selection of the numerical values (or symbols) representing bindings in a network. as will be detailed.

it on (see Figure 1). [7]

Take the example used before:

$$grow(x, y) \longrightarrow agr(x)$$

and the input is $grow(a, b)$, where a and b are constants. We will have two assemblies: one for $grow(x, y)$ and the other for $agr(x)$. The first assembly has three nodes: C, X, and Y. The second assembly has two nodes: C and X. The C node in the first assembly is linked to the C node in the second assembly. The X node in the first assembly is linked also to the X node in the second assembly (see Figure 2). [8] First the C node in the first assembly is activated, and the binding a is passed to the X node in grow(x,y), and b to the Y node. Then by applying the rule (i.e., propagating activation along the link), the C node in the second assembly is activated, and a is passed from the X node in $grow(x, y)$ to the X node in $agr(x)$. Thus a conclusion is reached: $agr(a)$.

The above discussion regarding how to add variables into the architecture is rather informal. One question that needs to be answered is exactly what functions we must have in a processing element (node) in order to deal with variables. We thus need some formal treatments that can qualify the problem clearly, as what follows.

3.2 A Formal Model of Node Activations

We will present here formal definitions for the basic components, namely *nodes*, *assemblies* and *networks*, in light of what was discussed above. First a new model of nodes is presented to subsume the weighted-sum node model, and assemblies are defined thereafter. Finally, we discuss how these elements can be used in variable binding.

[7] Each of the variable nodes in an assembly should also communicate with the C node to send its bindings, so that the C node can check consistency and others, and should receive "instructions" from the C node. The "instructions" are to tell a variable node which binding to take in case there are multiple inputs for a variable node. More details later.

[8] There should also be some intra-assembly links, which are omitted for simplicity. More on this later.

In order to incorporate variables and bindings, we will have to extend the definition of a *node* in conventional connectionist models. We will use a very general formalism, the *Discrete Neuronal Model*, or the DN model. [9] It can deal with variable binding, rule matching, and confidence value propagation all at the same time. Thus, the model forms a reasonable basis, and a unifying framework, for developing a complex reasoning system incorporating variables.

An automaton-theoretic description of a node in a DN model is a 7-tuple:

N=$< S, A, B, I, O, U, V >$
S= the set of all the possible states of a node,
O= the set of all outputs,
A= the set of all the actions (output symbols),
I= the set of inputs,
B= the set of all input symbols,
U= state transition function,
V= action function.

We think of a node as having a set of inputs and a set of outputs:

$$I = (I_1, I_2, ..., I_n) \in B'$$

$$O = (O_1, O_2, ..., O_m) \in A'$$

defining

$$B' =_{\text{def}} \Pi B$$

and

$$A' =_{\text{def}} \Pi A.$$

And the state transition and action functions are

$$U : S \times B' \longrightarrow S$$

[9]This formalism is first proposed in Sun [22] and later further developed in Sun [27] and Sun & Waltz [28]. It is a generalization of conventional connectionist models, aiming at resolving some difficulties inherent in these models (such as variable binding) by removing some unnecessary restrictions. The generalization in this formalism goes along several dimensions: internal states, differentiated outputs, and temporal responses. See also Aleksander [2].

$$V : S \times B' \longrightarrow A'$$

A network of DN nodes is a graph with (multiple) directed edges between pairs of nodes. An added constraint is if there is a directed link from node i to node j, then $A'_i = B'_j$. Although the formalism is complex (it is aimed for generality beyond this work; cf. Sun [22], [23], and [27]), the *actual* mechanism for variable binding is very simple, as will be seen below.

Because of the generality, this model can emulate other more conventional connectionist models (see Sun [24] for details). In case of emulating a model with continuous activation, V will be a function emulating the activation function in the original model (see [24]). U and S are not needed for this purpose. Conversely, DN nodes can also be implemented with the conventional weighted-sum threshold model. The question here is how to implement the output and the state transition function of a DN node. It has been shown that a multi-layer network can do the job: a network represents the current state (see Figure 3), and another network performs the mappings from the current input and state to the new state and output. The state value is fed back into the input layer of the network to help decide, together with the current input, which state to enter next (for details, see [24]). These two points together ensure the compatibility of this formalism with more conventional models.

We will describe now how the DN formalism can be used to construct a network, given some first order logic rules with variables. We can map from the first-order rules to a localist connectionist network (we call it CL) with variable binding as follows: A fact (a predicate) is represented by an assembly of DN nodes in CL, and rules are encoded in a network by the connections an assembly has to the other assemblies. For example, Figure 4 shows how assemblies representing facts are wired into a network. So CL is made up of assemblies which in turn are made up of nodes representing predicates and arguments, that is,

$$CL =_{\text{def}} \{AS, L\}$$

where

$$L =_{\text{def}} \{(AS_1, AS_2, i, j) \mid AS_1, AS_2 \in AS, \;\; i, j \in [0, k] \;\text{ are labels}$$

denoting nodes within an assembly}

$$AS =_{\text{def}} \{(C, X_1, X_2, ..., X_k) | \text{ k is an integer number}\}$$

where L is the connection patterns between assemblies, AS is a set of assemblies, C is the C node described before, and X's are the X nodes described before. The connections among nodes are specified in L by labels which indicate which node in one assembly is connected to which node in another assembly. There may also be connections from each X to C and from C to each X, as will be explained.

In each assembly, there are k+1 nodes of the DN type: C, $X_1, X_2, ..., X_k$. The C node contains the value of the fact predicate (the confidence value) and connects to other C nodes in other assemblies for representing different facts that are related to it by rules. The X nodes take care of variable bindings (i.e., for arguments to the fact predicate) for a maximum of k variables. The set of input/output symbols in the variable nodes are values (numeric or symbolic) representing all possible bindings in a task domain. The set is fixed a priori. Each element in an assembly, C or X's, is a node with a specific action function (a state transition function is actually not need for this application), as specified below:

For C:

$$O_C^1 = V_C^1(I_1, I_2, ...I_l, O_{X_1}^2, ...O_{X_l}^2) \in A_C$$

$$O_C^2 = V_C^2(I_1, I_2, ...I_l, O_{X_1}^2, ...O_{X_l}^2) \in A_C$$

where action function V_C^1 is a weighted-sum function (weights will be discussed later); V_C^2 is a signal to the X nodes (see examples later); $I_i's$ are outputs of other C nodes in other assemblies impinging on this node. A_C is the set of output symbols in the node C; Superscripts denote different outputs, where 1 is the inter-assembly output and 2 is the intra-assembly output; Node C also receive inputs from X nodes in the same assembly, useful in certain situations which will be explicated later.

And for X_i: i=1,2,....k,

$$O_{X_i}^1 = V_{X_i}^1(I_1, I_2, ...I_n, O_C^2) \in A_{X_i}$$

$$O^2_{X_i} = V^2_{X_i}(I_1, I_2, ...I_n) \in A_{X_i}$$

where action function $V^1_{X_i}$ is a mapping specifying the output of X_i based on I_i's, inputs at that moment from X_i nodes in other assemblies; $V^2_{X_i}$ is a signal to the C node (see examples later); A_{X_i} is the set of output symbols in the node X_i.

There are some subtle timing considerations that need to be addressed. Outputs of a node depend on not only inputs from other assemblies, but also inputs from other nodes in the same assembly, and thus it is important to make sure that there is no mutual waiting (deadlock) situation. In this case, intra-assembly communication can finish first, and then outputs to other assemblies will then be computed without any further impediments.

3.3 An Example

As an illustration, we will derive, in the DN formalism, a representation of a reasoning process for an example expressed in the first order logic rules. Let us look at the following example: proving that Socrates is mortal, based on the general knowledge: if one is human, one is mortal, and the knowledge that Socrates is human. The reasoning goes like this:

if one is human, one is mortal.
Socrates is human,
so Socrates is mortal.

The rule to be represented is

$$human(x) \longrightarrow mortal(x)$$

First we will set up a network for representing the rule. Applying the DN formalism, we build up two assemblies (see Figure 5). Each assembly contains a C node and a X node (a variable node for an argument). The first assembly represents "X is human" and the second assembly represents "X is mortal". and the input i is the value of the fact "Socrates is human", which is one in this case.

The output u for this case is the string "Socrates" or its numerical representation, and the output v is the value of the fact "Socrates is mortal", which is also one in this case.

For C1 (see Figure 5), let R stand for real, and Σ stand for string.

$$I = \{i, l\}$$

$$O = \{k, q\}$$

$$S = \Sigma$$

where B_i = R, B_l = Σ, A_k = R, and A_q= R. In this example, we assume that bindings are done by strings. However, in implementation, they can be easily replaced by numbers (activation values).

The signals from variable nodes to the C node are to inform it if there is a right binding present at each variable node, as mentioned above. The signals from the C node to variable nodes are to tell them if they should pass on bindings. In case the C node is not activated, the variable nodes in the same assembly should not, or need not, work on the variable binding. Although these issues are not important for this simple example, we will still keep these two types of links there for the sake of generality.

Now the state transition function and action function for C1:

$$U : R \times (R \times \Sigma) \longrightarrow R$$

$$V : R \times (R \times \Sigma) \longrightarrow A_k \times A_q = R^2$$

Expressing them in functional forms, we have

$$U(S, (i, l)) = U(i, l) = i$$

and

$$V(S, (i, l)) = V(i, l) = (k, q) = (i, l)$$

For X1,

$$I = \{j, k\}$$

$$O = \{l, p\}$$

$$S = \Sigma$$

where $B_j = \Sigma$, $B_k = \mathrm{R}$, $A_l = \Sigma$, and $A_p = \Sigma$. The state transition function and action function are

$$U : \Sigma \times (\Sigma \times R) \longrightarrow \Sigma$$

$$V : \Sigma \times (\Sigma \times R) \longrightarrow \Sigma^2$$

Expressing them in functional forms, we have

$$U(S, (j, k)) = U(j, k) = j$$

and

$$V(S, (j, k)) = V(j, k) = (l, p) = (i, (i \text{ if } l \neq 0, \quad \perp if\ l = 0))$$

where $\perp$ is a special symbol, indicating no output.

It is the same for C2 and X2. In case of C2:

$$U(S, (q, r)) = U(q, r) = q * c$$

where c is a weight (c=1).

$$V(S, (q, r)) = V(q, r) = (u, s) = (q * c, q)$$

And for X2:

$$U(S, (p, s)) = U(p, s) = p$$

$$V(S, (p, s)) = V(p, s) = (v, r) = ((p \text{ if } s \neq 0, \quad \perp otherwise), p)$$

In this states and state transitions are not needed, as demonstrated by the above equations. In the following discussion, we will no longer be concerned with states.

Simplifying the above notations, we come up with simple assignment statements (so that they will be easily programmable) as follows (assume prime signs represent the next time period; S is ignored here since it is not needed). For C1,

$$q' = i$$

$$k' = i$$

For X1,

$$l' = j$$

$$p' = j \text{ if } k' \neq 0, \perp \; otherwise$$

For C2,

$$u' = q * c$$

$$s' = q$$

For X2,

$$r' = p$$

$$v' = p \text{ if } s' \neq 0, \; \perp \; otherwise$$

Based on this, the whole process of reasoning can be described as follows:

$$u'' = q' * c$$

and

$$v'' = p' \text{ if } s'' \neq 0, \; \perp \; otherwise$$

And according to the above equations regarding p', q' and s', we have

$$v'' = j \text{ if } i \neq 0, \; \perp \; otherwise$$

$$u'' = i * c$$

where u is the output from the C node and v is the output from the variable node, all in the "mortal" assembly. In this case, the input i is 1, and the input j is "Socrates". So the output u is 1, and the output v is "Socrates", which together mean that "Socrates is mortal".

From the above discussions and the examples shown, it is clear that this solution to the variable binding problem has the advantage of being structurally simple, while at the same time maintaining the simplicity of processing elements (nodes). It compares favorably with other solutions in this regard.

4 Discussion

The approach develped in this paper can better deal with some difficult issues. We will look into these issues briefly. We will also look into the correctness of this solution in terms of logic.

4.1 Dealing with Some Issues

First of all, there is the issue of **multiple rules** reaching the same conclusion (with different activation values and possibly with different bindings), i.e., the question of how we combine conclusions from different rules. For example:

$$a(m,x)\ b(n,x) \longrightarrow c(m,n)$$

$$a(m,x)\ d(x,n) \longrightarrow c(m,n)$$

If both rules are activated, we have two conclusions — suppose e.g. $c(m,n)$ with a confidence value 0.9, and $c(m,n)$ with a confidence value 0.8. The question is how we should combine them (or choose one of them). In Touretzky & Hinton [29] the system picks one arbitrarily, and thus the result is unpredictable. This is certainly unsatisfactory. In this work, we will choose the one with the maximum activation value, or with regard to FEL, with the highest confidence value. This seems to be a reasonable choice, given the fact that we do not have any facility for recursion, such as stacks or recurrent connections. It is also the best choice when considering the rule interaction issue (which is beyond the scope of this paper; see Sun [24]). Nevertheless, we lose completeness by doing that, because the other conclusions with different bindings and with less than maximum activations are being disregarded (more on completeness later).

Second, the issue of **selection** follows: how do we choose and pass on the right binding in case there are multiple sets of bindings available? Existing solutions do not deal with this problem. With the DN model, we do the following: the C node (in the assembly representing the conclusion) receives inputs from different rules and MAX them (i.e. select the maximum), and will output k if its kth input is the maximum. Then in the corresponding X nodes (in the same assembly), if the input from the C node (intra-assembly input) is k, it will output the kth inter-assembly input. This simple two-step procedure basically takes care of the issue; some additional details are in Sun [24].

It should be pointed out that X nodes do not necessarily contain a particular numerical value or symbol that represents a particular domain object (constant);

in other words, they can be free variables. For example, if we have a rule:

$$p(x) \longrightarrow q(x)$$

given p(z), we can derive q(z), by representing z with a special value in the X nodes of the assemblies for p(x) and q(x). In order to distinguish a constant from a variable (or pseudo-binding), variables are represented by values greater than a certain threshold θ, so a simple test (such as "if $I_i > \theta$") suffices to distinguish them.

Now let us look at the issue of **consistency checking**. In order to check input bindings, all X nodes have to pass on their inputs to a central location, namely the C node, for comparisons. The C node then performs the equality or inequality checking, such as in the rule:

$$a(x_1, x_2)\ b(x_2, x_3) \longrightarrow c(x_1, x_2)$$

where the second argument in a and the first argument in b should be the same; or

$$a(x_1, x_2)\ \ x_1 \neq x_2 \longrightarrow c(x_1, x_2)$$

where the inequality must be enforced; another example is the identity checking, e.g.,

$$a(x, n) \longrightarrow c$$

when given the input $a(m, n)$, we want to make sure that the second argument of a is bound to n. These checkings can be easily formalized in the DN formalism because of its generality. For details, see Sun [24].

Another issue is that of **unification**, a much restricted version of the unification problem in first order logic. It is much restricted, because we do not allow functions in arguments. We can perform unifications of variable/variable, variable/constant, etc. In the above example

$$a(x, n) \longrightarrow c$$

if given input $a(m, y)$, how should we deal with the difference between n and y? (By the way, variables like x and y are represented by values greater than a certain threshold θ.) The answer is that we should unify n with y. This can be handled within the C node: if $y = n$ or if $y > \theta$, then unification can be done; otherwise, inputs are deemed inconsistent with the rule and further activations are blocked. What is especially difficult is the situation in which unification has to be done across predicates, as illustrated by the following example:

$$a(x_1, x_2)b(x_2, x_3) \longrightarrow c(x_1)$$

Suppose now we are given the following conditions:

$$a(m, n)$$

and

$$b(x_4, p)$$

where m, n, and p are constants; then we have to make sure that x_4 is also bound to n, since the second argument of a, which is the same as x_4, is bound to n. This can also be formalized in the DN formalism.

Yet another issue is **binding generation**. Let us look into the following example:

$$a(x_1) \longrightarrow b(x_1, x_2, c)$$

Two bindings here come out of the blue and have to be generated: x_2 and c. The binding generated for c is a genuine binding but the binding generated for x_2 is a *pseudo-binding*. Both cases can be taken care of in the assembly representing the conclusion of the rule, in the following way:

> In the corresponding X node, if input from the C node is 1, then output c; otherwise output $p > \theta$ for pseudo-bindings).

Let us now summarize what is *actually* needed in a node. Based on the previous discussions, types of common computation in a node include the following:

- Weighted-sum,
- MAX,
- Equality,

- Conditionals,
- Selection,

See Sun [24] for information regarding how they can be utilized to fulfill all the requirements of variable binding.

4.2 Correctness

Assume that there is no recursion whatsoever (i.e. hierarchical rule sets), the implementation with DN is complete and sound with respect to FEL; that is, whatever can be derived in FEL can be derived in the corresponding connectionist network, and vice versa. The implementation is also sound and complete in the binary case with respect to first order Horn clause logic. The completeness comes under the assumption that we replicate and rename a predicate (and all rules involving that predicate) throughout the entire rule set if it appears more than once with different arguments in RHS of rules. Otherwise, the outcomes from these different rules will be MAXed, and therefore the implemented system loses its completeness.

With recursive rules, either direct or indirect, the system will lose its completeness, if implemented with cyclic connections. However, we are mainly interested in the hierarchical FEL (as a comparison, note that human reasoning is mainly forward-chaining and does not involve circular inferences), so this is not a problem. Besides, there are also a number of ways we can circumvent recursions. One way is to replicate the predicate a number of times in the network, up to the maximum expected depth of the recursion, provided that we have that information a priori. Another possible way is to use an external stack, which stores and pops out multiple bindings when necessary.

It is interesting to note that in case of propositional FEL, no matter whether there is recursion or not, the implementation is always sound and complete. This is because there is no multiple binding problem at all for propositional logic.

5 Concluding Remarks

This paper discusses the problem of performing logical reasoning in connectionist networks beyond the functions as associative memories. A logic FEL is defined, which is a direct reformulation of connectionist networks and can handle Horn clause logic and Shoham's logic as special cases, demonstrating the logical capability of connectionist models. We present the solution to the variable binding problem. Formal definitions of DN nodes and assemblies are introduced and a connectionist architecture is defined to accommodate mechanisms for variable binding. We analyze carefully the solution to see how it can deal better with some difficult issues arising from existing solutions. This solution to the variable binding problem achieves the goal of structural simplicity while maintaining computational efficiency in individual processing elements.

6 References

[1] V. Ajjanagadde and L. Shastri, Efficient Inference with Multi-place Predicates and Variables in a Connectionist System, *Proc.11th Cognitive Science Society Conference*, pp.396-403, Lawrence Erlbaum Associates, 1989

[2] I. Aleksander, The logic of connectionist system, in: I. Aleksander ed. *Neural Computing Architectures*, MIT Press, 1989

[3] J. Barnden, The right of free association: relative-position encoding for connectionist data structures, *Proc.10th Conference of Cognitive Science Society*, pp.503-509, Lawrence Erlbaum Associates, 1988

[4] G. Blelloch, AFL-1: A Programming Language for Massively Parallel Computers, Master's Thesis, MIT AI Lab, 1986

[5] B. Buchanan and E. Shortliffe,(eds.) *Rule-Based Reasoning: the Mycin Experiment*, Addison-Wesley, 1984

[6] A. Collins and R. Michalski, The Logic of Plausible Reasoning: A Core Theory, *Cognitive Science*, Vol 13, No 1, pp.1-49, 1989

[7] M. Derthick, Mundane reasoning by parallel constraint satisfaction, TR CMU-CS-88-182, Carnegie-Mellon University, 1988

[8] C. Dolan and P. Smolensky, Implementing a connectionist production system using tensor products, In D. Touretzky et al. eds. *Proc.1988 connectionist Summer School*, pp.265-272. San Mateo, CA: Morgan Kaufman, 1989

[9] J. Feldman and D. Ballard, Connectionist models and their properties, *Cognitive Science*, pp.205-254, July 1982

[10] J. Fodor and Z. Pylyshyn, Connectionism and Cognitive Architecture: A Critical Analysis, in: Pinker and Mehler (eds.) *Connections and Symbols*, MIT Press, 1988

[11] S. Gallant, Connectionist Expert Systems, *Communication of ACM*, V.31(2), pp.152-169, 1988

[12] V. Girotto and P. Legrenzi, New European Contributions to the Psychology of Reasoning, *GOLEM*, pp.103-118, 1990

[13] P.J. Hayes, In defence of logic, *Proc.5th IJCAI*, pp.559-565, Morgan Kaufman, 1977

[14] B. Kosko, Fuzzy Cognitive Maps, *International Journal of Man-Machine Studies*, 24, pp.65-75, 1986

[15] T. Lange and M. Dyer, Frame selection in a connectionist model, *Proc.11th Cognitive Science Conference*, pp. 706-713, Lawrence Erlbaum Associates, 1989

[16] N. Nilsson, *Principle of Artificial Intelligence*, Tiogo, 1980

[17] S. Pinker and A. Prince, On Language and Connectionism, in: S. Pinker and J. Mehler (eds.), *Connections and Symbols*, MIT Press, 1988

[18] L. Shastri, A Connectionist Approach to Knowledge Representation and Limited Inference, *Cognitive Science*, 12, pp.331-392, 1988

[19] L. Shastri and V. Ajjanagadde, From simple association to systematic reasoning, Tech Report MS-CIS-90-05, University of Pennsylvania, 1990

[20] Y. Shoham, Reasoning about Change: Time and Causation from the standpoint of Artificial Intelligence, Ph.D Dissertation, Computer Science Department, Yale University, 1987

[21] P. Smolensky, On variable binding and representation of symbolic structure, Tech Report, University of Colorado, Boulder, CO, 1987

[22] R. Sun, A discrete neural network model for conceptual representation and reasoning, *Proc.11th Cognitive Science Society Conference*, pp.916-923, 1989

[23] R. Sun and D. Waltz, Neural networks and human intelligence, *Journal of Mathematical Psychology*, Vol.34, No.4, pp.483-488, 1990

[24] R. Sun, Integrating Rules and Connectionism for Robust Reasoning, Ph.D Thesis, Brandeis University, 1991

[25] R. Sun, Connectionist Models of Rule-based Reasoning, *Proc. 13th Cognitive*

Science Conference, pp.437-442, 1991

[26] R. Sun, Chunking and Connectionism, *Neural Network Review*, Vol.4, No.2, pp.76-78, 1991

[27] R. Sun, The Discrete Neuronal Models and the Discrete Neuronal Models, in: B. Soucek ed. *Neural and Intelligent System Integration*, pp.161-178, John Wiley & Sons, 1991

[28] R. Sun and D. Waltz, Neurally Inspired Massively Parallel Model of Rule-Based Reasoning, in: B. Soucek ed. *Neural and Intelligent System Integration*, pp.341-381, John Wiley & Sons, 1991

[29] D. Touretzky and G. Hinton, Symbols among neurons, *Proc.9th IJCAI*, pp.238-243, Morgan Kaufman, 1987

[30] L. Zadeh, Fuzzy Sets, *Information and Control*, 8, pp.338-353, 1965

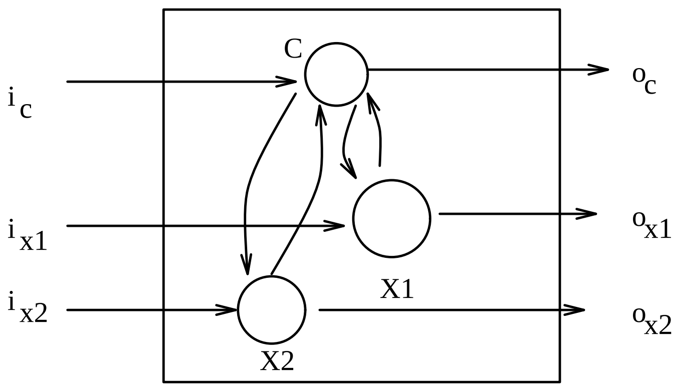

Figure 1: An Assembly

-

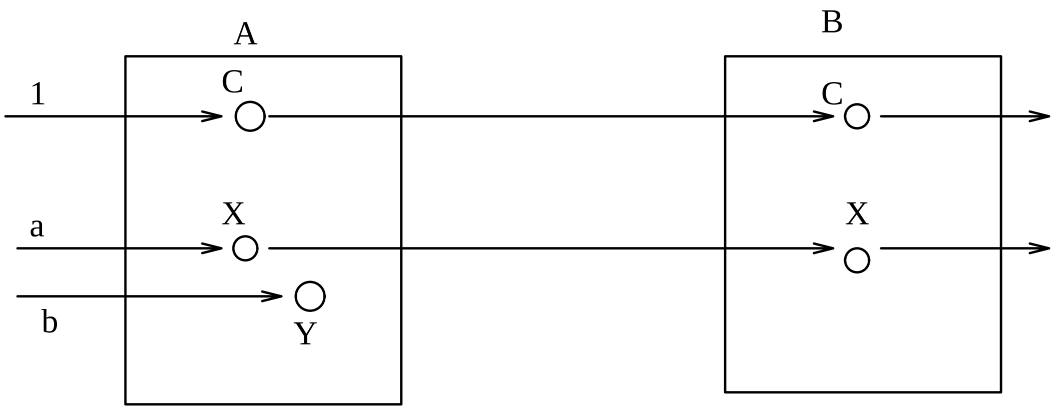

Figure 2: The Network for Representing the Rule: if grow(x,y) then agr(x)

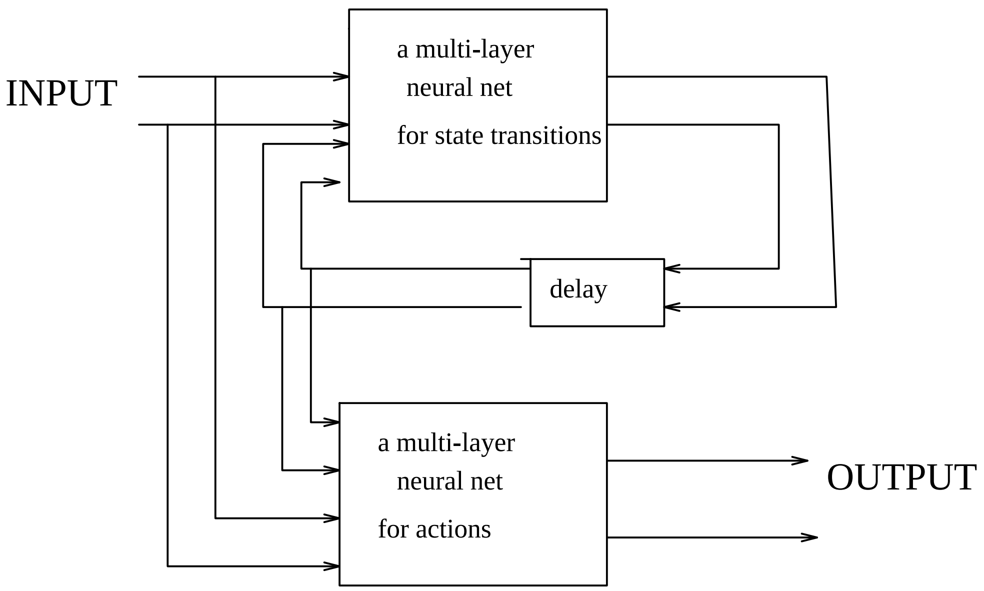

Figure 3: Implementing a Finite State Automaton in Weighted-sum Threshold Nets

-

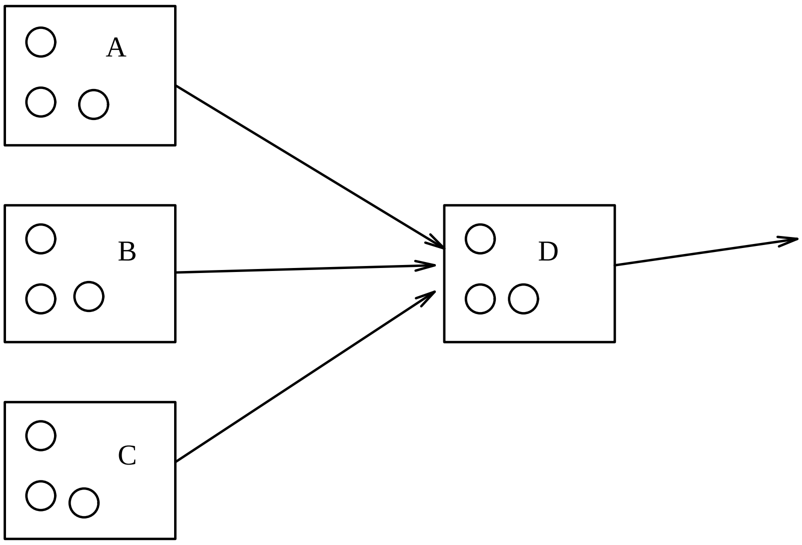

Figure 4: The Wiring of a Network for Representing the Rule: if A(X) B(Y) C(Z), then D(W)

where X, Y, Z are vectors of arguments.

-

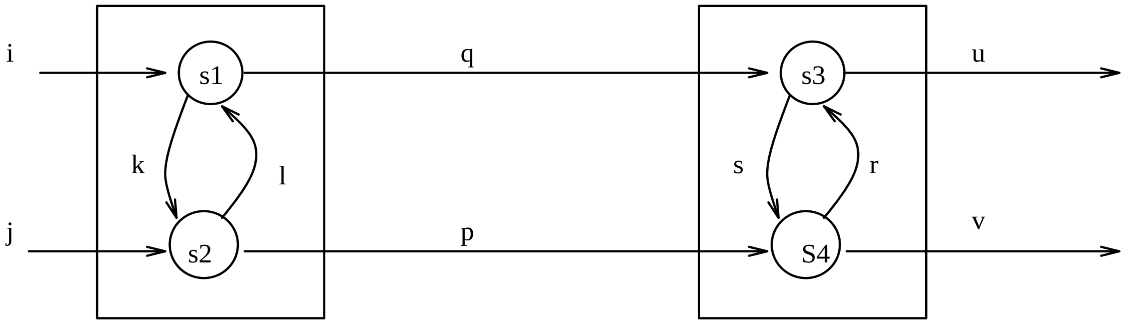

Figure 5: An Example Network — Socrates