# Symbolic Knowledge Encoding Using a Dynamic Binding Mechanism and an Embedded Inference Mechanism

Nam Seog Park, Dave Robertson, Keith Stenning
The University of Edinburgh

ABSTRACT

This chapter describes how synchronous activity between neuron elements can be used to build a dynamic binding mechanism and knowledge encoding mechanisms in a connectionist manner. The purpose of these mechanisms is to build a connectionist inference architecture which can replicate common symbolic styles of inference. To build such an inference architecture, an extended temporal synchrony approach [Park, Robertson and Stenning (1995)] is used as a basic building block. This is a revision and extension of an approach to the dynamic binding problem in connectionist systems, proposed by Shastri and Ajjanagadde (1993). On top of this, we introduce a set of algorithms which gives us a means of compiling a class of symbolic rules into a uniform inference network called a structured predicate network. This is used as a connectionist knowledge encoding mechanism which encodes symbolic rules and supports very fast inference.

## 1 INTRODUCTION

The study of oscillations in neural systems has attracted much attention from many researchers. Engle, Koenig, Kreiter, Gray and Singer (1991) observed that neurons in different parts of the brain oscillate in synchrony in response to certain stimuli. One hypothesised function for such synchronous activity is to solve the binding problem in human perception. Some researchers already have shown such a possibility in visual perception [Horn, Sagi and Usher (1991), Hummel and Biederman (1992), Strong and Whitehead (1989), Wang and Terman (1995)].

Using temporal synchrony in a connectionist model to support inference is found in the early work of Fahlman (1979) in which temporal synchrony is represented by the notion of marker passing. Each node in the network stores a small number of discrete markers. These markers propagate between nodes under the supervision of the network controller to achieve dynamic bindings and symbolic inference. Thus, nodes in his system were required to have high computational abilities, in order to store, match, and selectively propagate maker bits. Later Clossman (1988) used synchronous activity to represent argument-filler bindings for a connectionist model of categorisation and learning, but without an efficient encoding mechanism for rules and facts. Recently Ajjanagadde and Shastri (1991) and Shastri and Ajjanagadde (1993) have proposed another connectionist model for reflexive reasoning called SHRUTI. It provides an efficient connectionist mechanism to represent dynamic bindings based on temporal synchrony and other mechanisms which encode rules and facts involving predicates with $n$ arguments. However, SHRUTI's dynamic binding mechanism cannot properly represent a variable bindings and Park et al. (1995) extended their dynamic binding mechanism to cope with this limitation.

The purpose of this chapter is to introduce a connectionist inference architecture which encodes symbolic rules into the corresponding networks and performs fast inference over them. To start with, we first describe in Section 2 a type of symbolic knowledge and common styles of symbolic inference that we would like to replicate using the proposed connectionist inference architecture. Then, Section 3 will be devoted to explain the dynamic binding problem in connectionist systems and to outline the temporal synchrony approach and its extension. The emphasis will be on how temporal synchronous activities among neuron elements can be used to support dynamic bindings. This section also deals with how sub-tasks involved in symbolic styles of inference can be represented in a connectionist manner. In Section 4, we propose a connectionist inference architecture which uses the extended dynamic binding mechanism and the connectionist equivalents to the symbolic inference sub-tasks. A rule encoding and a connectionist inference procedure by the proposed architecture will be demonstrated using some example rules. Section 5 compares our inference architecture to other similar inference models currently available. Finally Section 6 discusses the results and concludes this chapter.

## 2 A BASIC SYMBOLIC INFERENCE

Predicate calculus is a formally defined representation scheme which allows us to express the knowledge needed to solve a problem. Our chosen knowledge representation scheme is a subset of *first-order Horn clause expressions* which is a set of universally quantified expressions in first-order predicate calculus of the form:

$$p_1(...) \wedge p_2(...) \wedge ... \wedge p_n(...) \rightarrow q(...)$$

where $p_i(...)$ and q(...) are positive atomic expressions called *predicates*. The conjunction of $p_i(...)$ is called the *antecedent* and the q(...) the *consequent*. An expression with no antecedent is called a *fact* and an expression which has both antecedent and consequent is called a *rule*. The symbol $\rightarrow$ is normally read as "implies" and called the *implication*. Some example sentences translated into facts and rules are:

"Adam is a man.": *man(adam)*
"For any X, if X is a man then X is mortal.": $man(X) \rightarrow mortal(X)$
"John loves Mary.": *love(john,mary)*
"For any X and Z, if there is some Y such that X loves Y and Y loves Z then X is jealous of Z.": $love(X,Y) \wedge love(Y,Z) \rightarrow jealous(X,Z)$

As can be seen, the basic structured component of rules and facts is a predicate. Each predicate having $n$ arguments (n-ary predicate) is represented using a predicate symbol followed by $n$ terms where terms are constant symbols or variable symbols.

If we translate the statement "Every man is mortal. Adam is a man." into first-order Horn clause expressions, the first statement is represented by the rule

$$\mathrm{man(X)} \rightarrow \mathrm{mortal(X)}$$

based on the notion that "X is a man" by the predicate *man(X)*, and "X is mortal" by the predicate *mortal(X)*. The two symbols *man* and *mortal* are called predicate names and X's in the bracket are called *arguments* of the predicates.

The second sentence, "Adam is a man", is then represented by one predicate.

$$man(adam).$$

In order to draw inference based on these expressions, we need inference rules. An inference rule is a mechanical means of producing a new expression from other expressions. Two inference rules frequently used are *universal instantiation* and *modus ponens* [Luger and Stubblefield (1989)]:

- *universal instantiation* states that if any universally quantified variable in a true expression is replaced by any appropriate term from the domain, the result is a true expression. Thus, if $a$ is from the domain of X, the $\forall X$ *p(X)* lets us infer *p(a)*. The meaning of the symbol $\forall$ is "for all" and is called the *universal quantifier*.

- if the fact $p$ and the rule $p \rightarrow q$ are known to be true, then *modus ponens* lets us infer $q$.

Because the $X$ in the rule is universally quantified, we may substitute any value (also called a *filler*) in the domain for $X$ and it still has true statement under the inference rule of universal instantiation. Based on the known fact, *man(adam)*, we can substitute adam for $X$ in the rule. As a result, we obtain the expression, $man(adam) \rightarrow mortal(adam)$. From this new expression and the known fact, man(adam), we can now apply modus ponens and infer the conclusion *mortal(adam)*.

In order to apply the inference rule, modus ponens, an inference system must be able to determine when two expressions are the same or match. This requires a decision process for determining the variable substitutions under which two or more expressions can be made identical. In the example rule and fact, for instance, the antecedent of the rule *man(X)* and the fact *man(adam)* can be identical when we substitute *adam* for the variable $X$. This substitution is called a binding and represented by the notation {adam/X}. This binding is used later to substitute the value *adam* for all the occurrences of the variable $X$ in the rule to obtain a new expression on which we can apply modus ponens inference rule. In summary, a step of forward chaining inference based on modus ponens is carried out by two subtasks, *matching* and *substitution.* For a given set of rules already known, when a fact is presented, the matching sub-task tries to match between the presented fact and the antecedents of rules. If there is any match, it produces a set of bindings. The substitution sub-task then uses these bindings to substitute all occurrence of variables in the rule matched. The resulting expression is used to infer the conclusion of inference.

In some cases, a variable can be identified with another variable. For example, the sentence, "There is someone who is a man.", could be translated into the expression, *man(U)*, where $U$ is a variable standing for someone. Matching this fact with the antecedent of the example rule produces the binding, {U/X}. Substituting U for all X's in the rule generates the new expression, $man(U) \rightarrow mortal(U)$. By applying modus ponens, we reach the conclusion "$U$ is mortal.".

Therefore, according to types of symbol names involved, there are two types of bindings: a constant binding and a variable binding. The term *constant binding* will be used to refer to the situation where a variable is bound to a constant, and *variable binding* the situation where a variable is bound to a variable. For instance, a binding between the constant *adam* and $X$, {adam/X}, is called a constant binding, and a binding between the variable $U$ and $X$, {U/X}, a variable binding.

Any connectionist system which replicates this type of symbolic inference must provide ways of representing symbolic knowledge and inference rules as well as representing bindings in a connectionist manner. This requires us to represent in a connectionist style the basic components of symbolic rules, such as symbols and predicates having $n$ arguments. Representing inference rules requires how to represent the subtasks involved in a step of symbolic inference.

# 3 REPRESENTING DYNAMIC BINDINGS

## 3.1 Dynamic Bindings and Symbolic Inference

Let us consider the sentence, "If someone owns something, she or he can sell it.", which may be represented by the following rule:

$$own(X, Y) \rightarrow can\_sell(X, Y).$$

By being told the fact, "John owns Car21.", which may be translated into the predicate *own(john,car21)*, one can infer the conclusion, "John can sell Car21.". In the same way, by being told the fact, "Mary owns Book42.", one can infer the different conclusion, "Mary can sell Book42.", using the same rule. During these inference procedures, an inference system dynamically produces a different set of bindings between the arguments of the antecedent of the rule and their corresponding fillers to support each inference: {john/X, car21/Y} at the first time and {mary/X, book42/Y} at the second time. These dynamic bindings are used to generate new expressions

$$own(john, car21) \rightarrow can\_sell(john, car21),$$
$$own(mary, book42) \rightarrow can\_sell(mary, book42),$$

from which both conclusions are reached. The importance of a dynamic representation of bindings is therefore due to its ability to create a new temporal binding to apply inference rules.

However, the ability of a connectionist system to represent dynamic bindings has been doubted [McCarty (1988), Foder and Pylyshin (1988)]. because the expression "John owns Car21.", *owns(john,car21)*, cannot be represented dynamically by simply activating the nodes representing the arguments, *own:X* and *own:Y* and the nodes representing the fillers, *john* and *car21*. The problem is that the given expression does not merely express an association between the arguments and fillers, it expresses a specific relation in which each binding between an argument and a filler plays a distinct role in the expression [Shastri and Ajjanagadde (1993)]. What is required is a connectionist representation of correct bindings between predicate arguments and their fillers, {john/own:X, car21/own:Y}, from which we can recognise the expression, own(john,car21), dynamically.

### 3.2 A Temporal Synchrony Approach

One solution to this dynamic binding problem is a temporal synchrony solution proposed by Shastri and Ajjanagadde (1993). The basis of this solution is a separation of the period of the oscillation cycle into several phases and the use of phase-sensitive neuron elements. A *phase* is a minimum time interval in which a neuron element performs its basic computations – counting the number of spikes and comparing to a threshold. An *oscillation cycle* is a window of time in which neuron elements show their oscillatory behaviours.

The computational behaviours of the phase-sensitive neuron elements are as follows:

- r-btu elements become active on receiving one spike in an oscillation cycle. On becoming active, r-btu elements produce a train of spikes that are in phase with the driving inputs.

- t-or elements become active on receiving one or more spikes within an oscillation cycle. On becoming active, t-or elements produce an oscillatory pulse train whose pulse width spans the entire oscillation cycle.

- multi-phase t-or elements become active on receiving more than one inputs in different phases within an oscillation cycle. On becoming active, multi-phase t-or elements produce an oscillatory pulse train whose pulse width spans the entire oscillation cycle.

Their approach also uses a t-and node which behaves as a temporal AND node. However its computational behaviour is not discussed because this chapter will not need it for the purpose of presentation.

If one phase-sensitive element A is connected to another element B, a periodic activation of A leads to a periodic and in phase activation of B. It is assumed that a neuron element can respond in this manner as long as the period of activation (oscillation cycle), $\pi$, lies in the interval $[\pi_{min},\pi_{max}]$. This interval can be interpreted as defining the frequency range over which a neuron element can sustain a synchronised response. A threshold, $n$, associated with each element indicates that the element will be active only if it receives $n$ or more synchronous inputs. Fig. 1 graphically depicts neuron elements and their temporal characteristics.

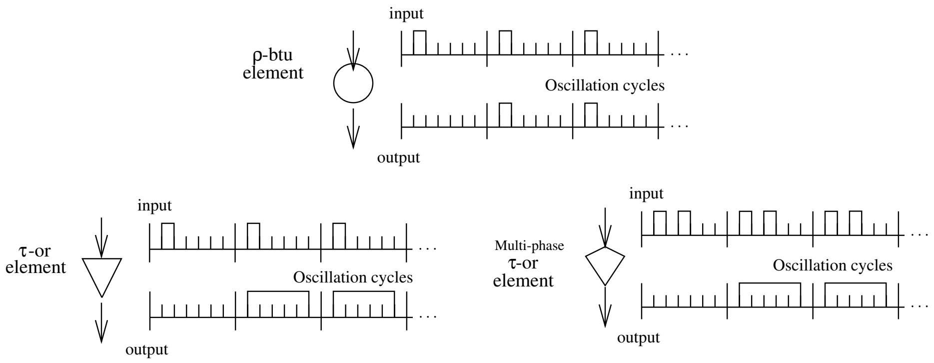


**Fig. 1** Temporal behaviours of phase-sensitive neuron elements

The temporal synchrony approach uses these neuron elements to represent components of symbolic rules to build networks encoding them. Arguments of expressions and potential fillers for those arguments are represented using r-btu elements, and an n-ary predicate in a rule is represented by an assembly of $n$ r-btu elements, where each element corresponds to each argument. The predicate *p(X,Y)*, for instance, is represented by using an assembly of two r-btu elements corresponding to the arguments *p:X* and *p:Y*. Fig. 2a depicts this predicate and the two constant fillers *a* and *b*.

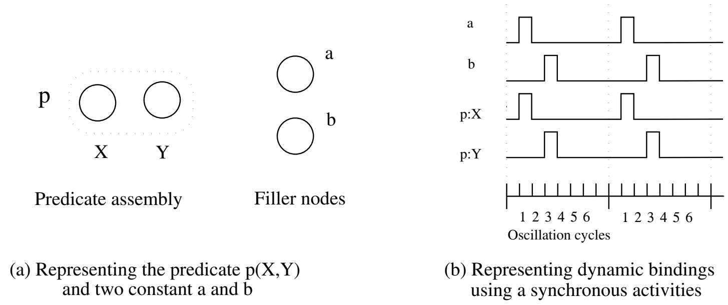


**Fig. 2** A representation of the predicate and dynamic bindings

Based on this representation, the set of bindings, {a/p:X,b/p:Y}, which will be used to generate the new predicate *p(a,b)* is represented in the following ways:

- the constant binding between the argument *p:X* and the constant *a*, {a/p:X}, is represented by activating in the same phase the element representing *p:X* and that representing *a*. Similarly, the constant biding {b/p:Y} is represented by activating in the same phase the element representing *p:Y* and that representing *b*.

Fig. 2b demonstrates these two constant dynamic bindings. Another set of dynamic bindings, {c/p:X, d/p:Y}, to produce the new predicate *p(c,d)* can also be represented simply activating two r-btu elements representing the new fillers, *c* and *d*, in phase with the elements representing arguments, *p:X* and *p:Y*. Thus, the temporal synchrony approach can represent dynamic bindings very efficiently by using synchronous activity between neuron elements.

This solution, however, has limitations [Park et al. (1995)]. One important limitation is that it cannot represent variable bindings. This is because r-btu elements are used to represent only constant fillers in their mechanism. For instance, it cannot differentiate two variable bindings, {U/p:X, V/p:Y} because no r-btu node is used to represent the variable fillers *U* and *V*. Consequently, the r-btu nodes representing *p:X* and *p:Y* remain inactive throughout inference. Connectionist systems which replicate symbolic inference should be capable of dealing with both constant and variable bindings dynamically without disrupting the structure of the network.

### 3.3 An Extended Temporal Synchrony Approach

Park et al. (1995) extended the temporal synchrony approach to overcome the limitation demonstrated in the previous subsection. This extension involves generalisation of r-btu elements and introduction of entity nodes.

#### 3.3.1 Generalisation of r-btu Elements

Shastri and Ajjanagadde's system restricted their r-btu element to carrying only one spike per oscillation cycle. We generalise the r-btu element so that it can carry more than one spike per oscillation cycle. The detailed temporal behaviour of the generalised element, which will be call a p-btu element, is as follows:

> A p-btu element becomes active on receiving one or more spikes in different phases of any oscillation cycle. On becoming active, a p-btu element produces oscillatory spikes that are in-phase with the driving inputs. A threshold, $n$, associated with each element indicates that the element will fire only if it receives $n$ or more spike inputs in the same phase.

Fig. 3 demonstrates the behaviour of p-btu nodes.

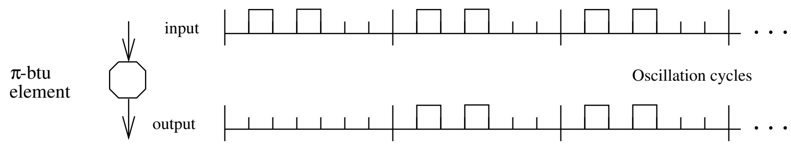


**Fig. 3** Temporal behaviour of a p-btu node

This p-btu node allows multiple spikes per cycle and propagates these in synchrony with the driving inputs. This requires higher signal transfer rates than Shastri and Ajjanagadde's r-btu nodes. Eckhorn (1993) observed that the signal transfer rates on biological neurons are much higher than those of r-btu nodes. Such high rates in a single neuron are capable of signalling complex messages, including routing dynamic representations. This allows the possibility that there exists a type of neuron whose signal transfer ability is even higher than the p-btu node.

### 3.3.2 Representing Entities and Predicates

To represent symbolic entities, the extended approach introduces an entity node. An entity node is a pair of p-btu elements. The left element is used to represent a variable role of the entity and the right element a constant role. A symbolic entity name may be used as a label of an entity node. For convenience in discussion, we represent an entity node using the notation, *entity_name([0],[0])*, where the left square bracket represents the state of the left p-btu element and the right square bracket that of the right p-btu element. The symbol "0" denotes that the element is inactive.

The notations, *entity_name*$_{left}$*[0]* or *entity_name*$_{right}$*[0]* are used to indicate each state of the left or the right p-btu element. When an entity node is used to represent a constant filler, only its right element is used. When the entity node is used to represent a variable filler, only its left element is used. However, when it is used to represent an argument of a predicate, either or both elements may be used at the same time. When an entity node is used simply to represent a constant or a variable filler, the notations, *var_name[0]* or *const_name[0]* is used to shorten the representation.

Using entity nodes, an n-ary predicate is then represented by an assembly of $n$ entity nodes, labelled $A_1$, $A_2$, ..., $A_n$, where the entity node $A_i$ represents the $i$th argument of the predicate. We use the notation,

$$pred\_name{:}\{A_1([0],[0]),\ A_2([0],[0]),\ \ldots,\ A_n([0],[0])\}$$

to represent an n-ary predicate assembly symbolically. Fig. 4a depicts a graphic representation of the predicate assembly representing $p(X, Y)$ and two fillers $a$ and $U$.

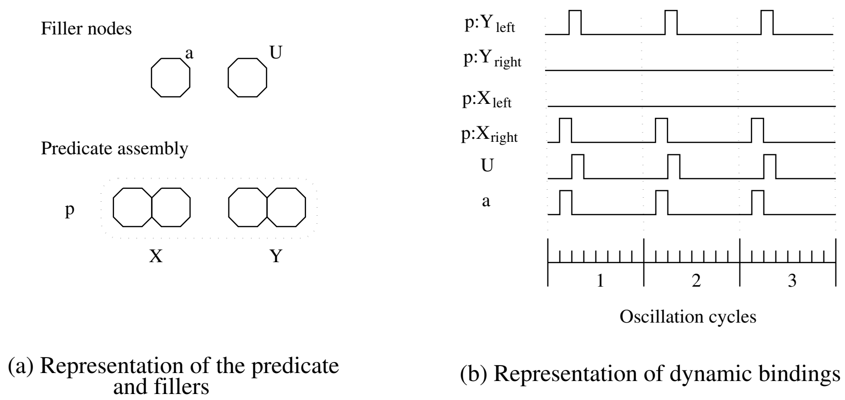


**Fig. 4** The extended representation of the predicate and dynamic bindings

### 3.3.3 Representing Dynamic Bindings

Since entity nodes are used to represent both fillers and arguments, dynamic bindings are represented using temporal synchronous activity between entity nodes. When the predicate $p(X, Y)$ and the two fillers $a$ and $U$ are represented using a predicate assembly and two entity nodes as shown in Fig. 4a, the dynamic bindings, {a/p:X, U/p:Y}, between these fillers and the arguments of the $p$ predicate assembly are represented in the following ways:

- the constant binding between $a$ and $p{:}X$ is represented by activating the entity node representing $a$ and the right element of the entity node representing $p{:}X$ in the first phase so that $a[0]$ becomes $a[1]$ and $P{:}X_{right}[0]$ becomes $P{:}X_{right}[1]$, where the number 1 stands for the first phase;

- the variable binding between $U$ and $p{:}Y$ is represented by activating the entity node representing U and the left element of the entity node representing $p{:}Y$ in a new phase, the second phase for example, so that $U[0]$ becomes $U[1]$ and $p{:}Y_{left}[0]$ becomes $p{:}Y_{left}[2]$, where the number 2 indicates the second phase.

Fig. 4b depicts the phasic representation of these dynamic bindings. Basically the in-phase activation of two or more entity nodes represents bindings between them. Constant bindings or variable bindings are distinguished by the position of the activated element of the entity node. This is a feature absent in Shastri and Ajjanagadde's temporal synchrony approach.

### 3.4 Representing Inference Procedures

Even if we have a connectionist binding mechanism and a way of representing symbolic knowledge components, there still remains the problem of how to represent symbolic inference procedures in a connectionist manner. As described in Section 2, a step of symbolic inference involves two sub-tasks, matching and substitution. If we reconsider the following rule again

$$own(X,Y) \rightarrow can\_sell(X,Y).$$

Presenting the fact "John owns something.", which may be represented by the predicate $own(john,U)$, to an inference architecture should result in the set of bindings {john/own:X, U/own:Y} to complete the matching sub-task. If the predicate $own(X,Y)$ and the two fillers, $john$ and $U$, are represented using the following predicate assembly and entity nodes

$$own{:}\{X([0],[0]),\ Y([0],[0])\},\ john[0],\ U[0]$$

the above situation can be represented easily by setting up dynamic bindings between argument nodes of the predicate assembly and the two filler nodes as follows:

$$own{:}\{X([0],[1]), Y([2],[0])\},\ john[1],\ U[2].$$

The in phase oscillations between $own{:}X_{right}[1]$ and *john[1]* and between $own{:}Y_{left}[2]$ and *U[2]* represent the required bindings, {john/own:X, U/own:Y}.

Once a set of bindings are obtained, the substitution sub-task substitutes these fillers for the arguments and this results in the new expression

$$own(john, U) \rightarrow can\_sell(john, U).$$

If the predicate *can_sell* is represented with the predicate assembly

$$can\_sell{:}\{X([0],[0]), Y([0],[0])\}$$

The effect of this substitute sub-task may be achieved in a connectionist manner by activating $can\_sell{:}X_{right}[0]$ in the first phase and $can\_sell{:}Y_{left}[0]$ in the second phase as follows:

$$can\_sell{:}\{X([0],[1]), Y([2],[0])\}$$

Then, the new oscillatory behaviour between the constant filler node *john[1]* and $can\_sell{:}X_{right}[1]$ represents the new constant binding {john/can_sell:X} and in phase activation between the variable filler node *U[2]* and $can\_sell{:}Y_{left}[2]$ the new variable binding {U/can_sell:Y}. From the sets of dynamic bindings obtained from the *own* and *can_sell* predicate assemblies, a new expression $own(john, U) \rightarrow can\_sell(john, U)$ can be obtained to conclude *can_sell(john, U)*.

Observation of these sub-tasks and their connectionist representations suggests that a step of inference based on a rule can be achieved in a connectionist manner by the following procedure:

Step1: when a rule is given, represent the antecedent and the consequent of the rule using predicate assemblies; The predicate assembly corresponding to the antecedent is called the antecedent predicate assembly and that corresponding to the consequent of the rule the consequent predicate assembly;

Step2: when a fact is presented, set up initial dynamic bindings between the fillers appearing in the presented fact and the argument nodes of the antecedent predicate assembly of the rule matched in the following way:

- assign each entity node (let us call it a filler node) to represent each filler appearing in the presented fact;
- activate each filler node and its corresponding argument node of the predicate assembly matched in a unique phase in such a way that: if the filler is a constant, activate both the right elements of the filler and the argument nodes; if the filler is a variable, activate both the left elements of the filler and the argument nodes.

Step3: to complete a step of inference, propagate the initial dynamic bindings from each argument node of the antecedent predicate assembly to its corresponding argument of the consequent predicate assembly (which share the same argument name).

The last step of inference is possible by setting up links between each argument node of the antecedent predicate assembly and the corresponding argument node of the consequent predicate assembly after the antecedent and the consequent of the rule are represented using their predicate assemblies. However, care must be taken when establishing these links because this automatic propagation of initial bindings from the antecedent predicate assembly to the consequent predicate assembly can be affected by the type of rules involved.

### 3.5 Handling Difficult Issues

Until now a simple rule has been used to illustrate the extended dynamic binding mechanism and connectionist symbolic inference procedures. If the proposed mechanism has to deal with a more complex symbolic rule such as $p(X, X) \rightarrow q(X)$ which requires repeated arguments of the $p$ predicate to get bound to the same constant filler or free variable fillers, an additional connectionist mechanism is needed to force this condition. This is because the initial binding instantiation mechanism simply sets up bindings without involving any consistency checking within a group of unifying arguments (the repeated arguments $X$'s for $p$). Therefore, an additional connectionist sub-mechanism is needed to detect a consistency violation when two different constant fillers are assigned to different occurrences of the same variable argument.

This requires an additional sub-mechanism to be added apart from the links which connect the antecedent predicate assembly with the consequent predicate assembly. The role of this additional sub-mechanism is to ensure consistency conditions that are enforced by the syntax of a rule and to block binding propagation from the antecedent predicate assembly to the consequent predicate assembly if there is any consistency violation.

Other types of rules which also require special treatment are

- rules having constant arguments: *p(a,b,X)* $\rightarrow$ *q(b,X)*; If constant arguments appear in the antecedent of a rule, an additional sub-mechanism has to perform consistency checking to ensure all the fillers assigned to the constant arguments are the same as the ones specified as constant argument names.

- rules requiring consistency checking across different groups of unifying arguments: $p(X, X, Y, Y) \rightarrow q(X, Y)$; This is more general case of the rule $p(X, X) \rightarrow q(X)$ but requires binding interaction across different groups of unifying arguments (between X's and Y's). Thus, it requires us to check any consistency violation occurring across these two groups of unifying arguments.

Since each type of rules requires a different structure for the additional sub-mechanism, special algorithms are needed to decide a proper structure of the sub-mechanism according to the type of the given rule. The algorithms for these tasks will be described in greater detail in the next section.

## 4 A CONNECTIONIST INFERENCE ARCHITECTURE

Having introduced the extended connectionist dynamic binding mechanism and the connectionist symbolic inference procedures, the rest of the chapter describes how these mechanisms can be used as basic building blocks for a connectionist inference architecture.

### 4.1 A Structured Predicate Network

When a symbolic rule is encoded, representation of the antecedent and the consequent is easily done by creating their corresponding predicate assemblies. The main issue is how to build a proper intermediate mechanism between them because the intermediate mechanism has to be designed not only to provide a path for binding propagation but also to prevent incorrect substitution during unification. This needs to take into account consistency of bindings in the antecedent; correct propagation of bindings from antecedent to consequent; and consistency of bindings in the consequent after binding propagation. This suggests that each rule needs to be encoded into a special type of network which has a unique intermediate mechanism to support a target form of symbolic inference. This special type of network, which will be called a structured predicate network (SPN), can be defined as follows:

**Definition:** A structured predicate network (SPN) is a localist network which encodes a rule. Each SPN is composed of three parts, $SPN = \{P_a, M, P_s\}$, where $P_a$ is a predicate assembly representing the antecedent, $P_c$ is a predicate assembly representing the consequent, and $M$ is the intermediate mechanism which connects them. Therefore, the overall structure of a SPN looks like the network shown in Fig. 5.

Whenever a rule is encoded, the intermediate mechanism, $M$, for the rule will be implemented in such a way that it includes several sub-mechanisms which carry out all the necessary sub-tasks to achieve the target symbolic inference.

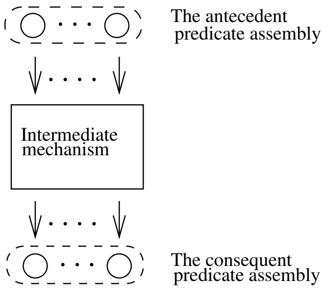


**Fig. 5** The structure of a SPN

## 4.2 Basic Definitions

This subsection defines some basic definitions that will be used to decide required sub-mechanisms when a rule is encoded.

**Definition:** For a given predicate, $p(A_1, A_2, ..., A_n)$, a *unifying argument group* (UAG) of the predicate $p$, $G_{A_i}$, is a set of arguments which have an identical symbol name $A_i$. If there are repeated arguments in the predicate, a series of subscript numbers may be used to differentiate one from others. If the argument group is for a constant name, $\mathrm{G}_{A_i}$ is called a *constant UAG* and if the argument group is for a variable name, $\mathrm{G}_{A_i}$ is called a *variable UAG*. Since each rule has an antecedent and a consequent, we obtain two sets of UAGs from a rule. To distinguish a set of UAGs generated by the antecedent from that produced by the consequent, we use two symbols, $\mathrm{S}_a$ and $\mathrm{S}_c$.

**Definition:** For two sets of UAGs corresponding to the antecedent and the consequent of a rule:

$$S_a = \{G_{A_1}, G_{A_2}, \ldots, G_{A_n}\}$$
$$S_c = \{G_{A_1}, G_{A_2}, \ldots, G_{A_m}\}$$

we say that $G_{A_i}$ in $S_a$ is *related* to $G_{A_j}$ in $S_c$ and vice versa if $A_i$ and $A_j$ refer to the same symbolic argument name, i.e. $A_i = A_j$. In other words, two UAGs obtained from the antecedent and the consequent are related if they are obtained from the same symbolic argument name.

To illustrate these definitions, let us consider the following rule:

$$p(X, X, Y, Y) \rightarrow q(X, Y)$$

From the antecedent, *p(X,X,Y,Y)*, we get

$$S_a = \{G_X, G_Y\}, \text{ where } G_X = \{p{:}X_1, p{:}X_2\},\ G_Y = \{p{:}Y_1, p{:}Y_2\}.$$

Note that the subscript numbers in the argument names of $G_X$ indicate the order of each argument repeatedly appearing in the predicate. This is simply to differentiate repeated arguments in different argument positions and does not affect the meaning of the original argument name.

From the consequent, *q(X,Y)*, we obtain:

$$S_c = \{G_X, G_Y\}, \text{ where } G_X = \{q{:}X\},\ G_Y = \{q{:}Y\}$$

We say that $G_X$ in $S_a$ is related to $G_X$ in $S_c$ because they are obtained from the same argument name $X$. Also $G_Y$ in $S_a$ is related to $G_Y$ in $S_c$.

**Definition:** The *size of* a set of UAGs is a number of UAGs in the set and the size of a UAG is a number of arguments in the UAG. If the size of a UAG is greater than 1, this means that the UAG was obtained from repeated arguments of a predicate.

If we consider the following rule

$$p(X, Y) \rightarrow q(X, Y, Z)$$

the $S_a$ and $S_c$ of the rule are

$$S_a = \{G_X, G_Y\}, \text{ where } G_X = \{p{:}X\},\ G_Y = \{p{:}Y\}$$
$$S_c = \{G_X, G_Y, G_Z\}, \text{ where } G_X = \{q{:}X\},\ G_Y = \{q{:}Y\},\ G_Z = \{q{:}Z\}$$

size($S_a$)=2 and size($S_c$)=3 and the sizes of all UAGs in $S_a$ and $S_c$ are 1.

4.3 Building An Intermediate Mechanism

When encoding each rule, a mechanical procedure is required to build the intermediate mechanism. This subsection describes such a procedure which determines the required sub-mechanisms automatically based on the information about two sets of UAGs ($S_a$ and $S_c$) obtained from the given rule.

### 4.3.1 Binding Collection For Each UAG

Any repeated arguments of the antecedent force the condition that they have to get bound to the same constant filler or free variable fillers during inference. This requires any bindings generated from the repeated arguments to be collected for consistency checking. A binding collection sub-mechanism (BCM) is needed for this purpose. The group of UAGs which need a BCM can be determined by checking the size of each UAG in the antecedent:

$$BCM = \{G_{Ai} | G_{Ai} \in Sa \wedge size(G_{Ai}) > 1\}$$

The structure of a BCM is demonstrated in Fig. 6. As can be seen, the BCM consists of one entity node, called a *binding node*, and associated links. The argument nodes representing the repeated arguments in $G_{Ai}$ are connected to the binding nodes through the links. When the initial bindings are set up on these argument nodes, they propagate to the binding node automatically. The left element of the binding node then represents variable bindings of all argument nodes and the right element constant bindings. The consistency of these intermediate bindings is checked later by another consistency checking sub-mechanism.

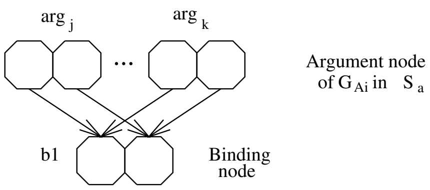


**Fig. 6** The structure of a BCM

4.3.2 Binding Propagation for Each UAG

To achieve symbolic inference, the initial bindings which are set on the antecedent predicate assembly needs to propagate to the consequent predicate assembly. Therefore, a binding propagation sub-mechanism (BPM) is needed to provide paths for binding propagation between the argument nodes of any UAG in the antecedent predicate assembly to those of the related UAG in the consequent predicate assembly. The UAGs which need a BPM are determined by the following condition:

$$BPM = \{G_{A_i} | \, G_{A_i} \in S_a \wedge G_{A_i} \in S_c\}$$

According to the type of a UAG, two different BPMs are required: A BPM for a variable UAG (BPM$_v$) and a BPM for a constant UAG (BPM$_c$). Fig. 7 illustrates them.

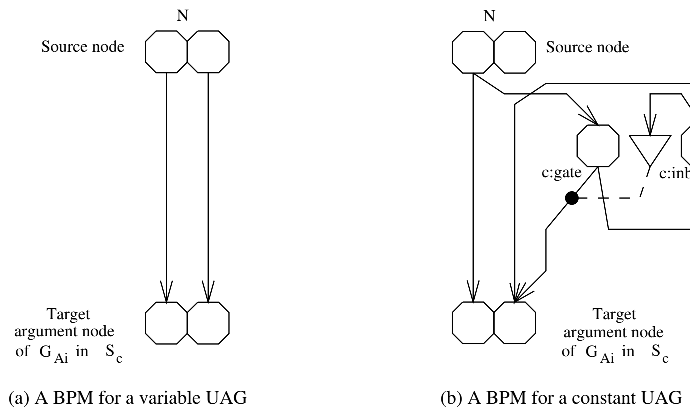


**Fig. 7** The structure of BPMs

In the figure, the source node $N$ is the binding node introduced as the BCM if the arguments of a UAG in $S_a$ are repeated ones. Otherwise, it indicates the argument node of the UAG in $S_a$. As can be seen in Fig. 7a, the BPM$_v$ for a variable UAG consists of simple links between the argument node of $G_{Ai}$ in $S_a$ and that of $G_{Ai}$ in $S_c$. However, the BPM$_c$ for a constant UAG is more complicated as shown in Fig. 7b. The constant argument of this example UAG is assumed to be $c$ for the purpose of explanation. If a variable filler is assigned to the source node of the $\mathrm{BPM}_c$, the variable binding propagates from $\mathrm{N}_{left}$ to the left element of the target argument node. At the same time, the same binding also propagates to the right element of the target argument node through the *c:gate* node to achieve unification between the variable filler and the constant argument. If the constant filler, $c$, is assigned to the source node, the right element of the target argument node receives the constant binding from the $c$ node directly. The inhibitory link (drawn in a dashed line) from the *c:inb* node is used to block the binding propagation path from $N_{left}$ to the right element of the target argument node through the *c:gate* node when the constant node $c$ becomes active at the initial binding setup stage.

### 4.3.3 Consistency Checking for Each UAG

Two types of consistency checking sub-mechanisms (CCMs) are required: constant consistency checking ($CCM_c$) for a constant UAG and variable consistency checking ($CCM_v$) for a variable UAG which has repeated arguments. Constant consistency checking forces the condition that all constant argument nodes in the UAG must get bound to the same constant specified as the constant argument or free variable fillers during inference. This condition should be forced whether or not the constant argument is repeated in the antecedent, i.e. regardless of the size of the UAG. In the same way, variable consistency checking forces all the repeated variable argument nodes in the UAG to get bound to the same constant filler or free variable fillers. The groups needing this treatment are determined by

$$CCM_c = \{G_{A_i} \mid G_{A_i} \in S_a \wedge const(A_i)\}$$
$$CCM_v = \{G_{A_i} \mid G_{A_i} \in S_a \wedge var(A_i) \wedge size(G_{A_i}) > 1\}$$

Fig. 8a illustrates a $CCM_c$ for the constant UAG which has one constant argument, $c$, and Fig. 8b a $CCM_v$ for a repeated variable UAG.

In Fig. 8a, the $CCM_c$ is represented using a t-or element labelled *to1*. This node will be activated whenever the right element of the source node is bound to a constant other than $c$. The activation of the *to1* node projects the inhibitory signal

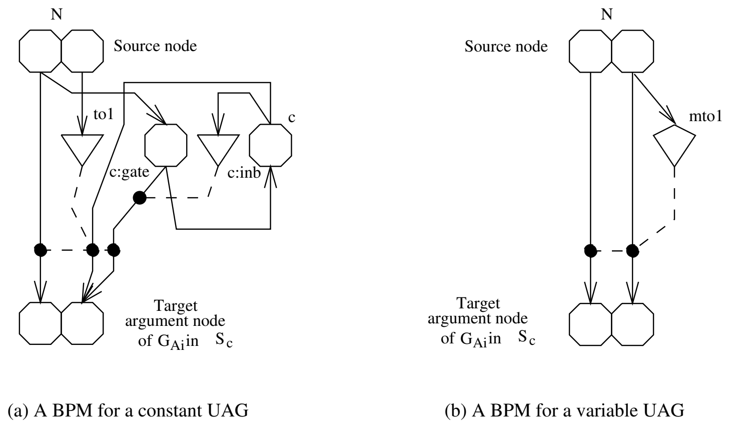


**Fig. 8** The structure of CCMs

which blocks the BPM between the source node and the target argument nodes. Note the dashed link from *to1* to one black dot near to the bottom of the network, which is connected to two other black dots by a dashed line. This is to abbreviate representation of a full connection from *to1* to each back dot. On becoming active, any node connected to one of these black dots send the same inhibitory signal to the rest of dots.

The $CCM_v$, by contrast, only needs the *mto1* node, which is a multi-phase t-or element (Fig. 8b). Whenever the right element of the source node receives two different constant bindings, the *mto1* node will detect this and projects the inhibitory signal to stop the flow of the activation from the source node to the target argument node.

### 4.3.4 Binding Interaction Between UAGs

For any pair of UAGs in the antecedent, if either or both of them are constant UAGs or repeated variable UAGs, binding interaction between them and consistency checking after binding interaction is required to ensure proper propagation of bindings from the source predicate to the target predicate. There are three combinations in pairing all the different types of UAGs in the proposed inference architecture:

- a pair of a constant UAG and a variable UAG
- a pair of two variable UAGs
- a pair of two constant UAGs

**Binding interaction between constant and variable UAGs:** For any two constant and variable UAGs, if the same variable filler is assigned to arguments belonging to them, the binding obtained between the variable filler and the arguments in the constant UAG must migrate to the arguments in the variable UAG . This situation can be seen when the fact *p(U,U)* is presented to the following rule:

$$p(c, X) \rightarrow q(c, X)$$

Presenting the fact first generates the bindings between the variable fillers ($U$'s) and the two arguments of the antecedent, which results in the set of bindings {U/p:c,U/p:X}. Since the arguments *p:c* and *p:X* are bound to the same variable filler, binding interaction between these two arguments is required during inference to produce the desirable result *q(c,c)* from the consequent of the rule.

The sub-mechanism which carries out this task is called a binding interaction sub-mechanism between a constant and a variable UAG ($BIM_{cv}$). The pairs of UAGs requiring this sub-mechanism are determined by the following condition:

$$BIM_{cv} = \{(G_{A_i}, G_{A_j})|\ G_{A_i} \in S_a \wedge G_{A_j} \in S_a \wedge const(A_i) \wedge var(A_j)\}$$

The structure of a $\mathrm{BIM}_{cv}$ is shown in Fig. 9a. The node *to1* is used to detect the situation where the same variable filler is assigned to the source node $N_i$ of a constant UAG and the source node $N_j$ of a variable UAG. Activation of the *to1* node projects the inhibitory signal which blocks direct binding propagation from the source nodes to the target argument node. The intermediate bindings generated as a result of binding interaction between two UAGs then will be represented on the binding node, *b2*. As inference continues, these bindings will propagate to the target argument nodes to complete inference. The *c:gate* is a node associated with the constant node *c*, and the *mto2* node is used for consistency checking after binding interaction.

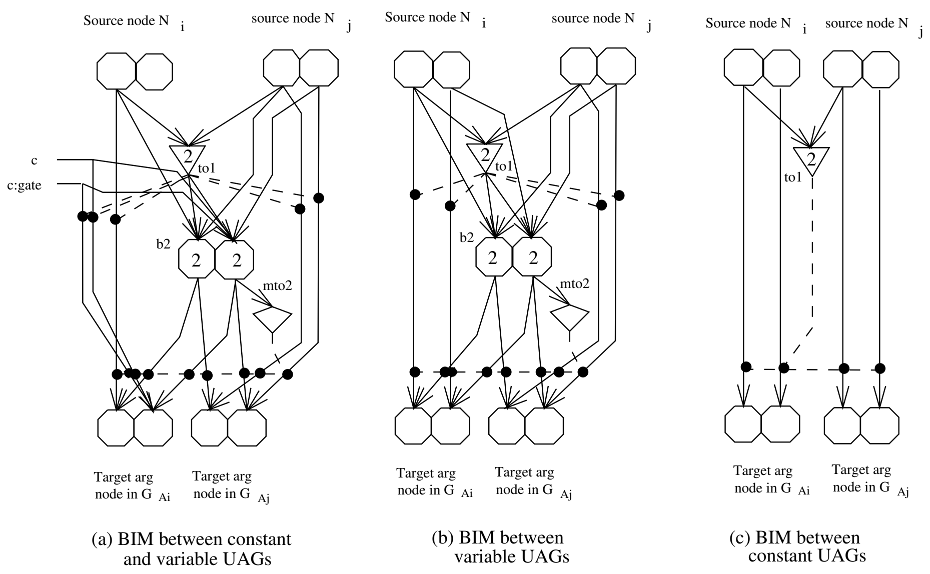


**Fig. 9** The structure of binding interaction sub-mechanisms

**Binding interaction between variable UAGs:** A binding interaction sub-mechanism between variable UAGs ($BIM_{vv}$) is similar to the $BIM_{cv}$ except the condition that at least one of UAGs should have repeated arguments rather than a single argument. This sub-mechanism is not needed between any two UAGs which have a single variable argument because no consistency checking is required in this case. The example rule which requires a $BIM_{vv}$ is

$$p(X, X, Y) \rightarrow q(X, Y).$$

Presenting the fact, *p(a,U,U)*, to the antecedent of the rule generates the set of bindings, {a/p:X,U/p:X,U/p:Y}, which require binding interaction between the repeated argument *p:X*'s and the third argument $Y$ during inference to produce the desirable result *q(a,a)*. Therefore, a sub-mechanism is needed for binding interaction between these two groups of unifying arguments.

The pairs of UAGs which require this sub-mechanism are determined by the following condition:

$$BIM_{vv} = \{(G_{A_i}, G_{A_j}) |\ G_{A_i} \in S_a \wedge G_{A_j} \in S_a \wedge var(A_i) \wedge var(A_j) \wedge A_i \neq A_j,$$
$$size(G_{A_i}) > 1 \ or \ size(G_{A_j}) > 1\}.$$

Fig. 9b illustrates the structure of a $BIM_{vv}$.

**Binding interaction between constant UAGs:** This refers the situation where the same variable filler is assigned to two different constant argument nodes as exemplified with the following rule:

$$p(a, b) \rightarrow q(a, b).$$

Presenting the fact *p(U,U)* should fail the matching between the presented fact and the antecedent of the rule because the same variable filler $U$ is bound to two different constant arguments $a$ and $b$. This indicates that for any two different constant UAGs in the antecedent, we need a binding interaction sub-mechanism ($BIM_{cc}$) which detects this situation and prevents the rule from firing during inference. These pairs of UAGs can be determined by the following condition.

$$BIM_{cc} = \{(G_{A_i}, G_{A_j})|\ G_{A_i} \in S_a \wedge G_{A_j} \in S_a \wedge const(A_i) \wedge const(A_j) \wedge A_i \neq A_j\}$$

Fig. 9c depicts the structure of a $BIM_{cc}$ between two source nodes, $N_i$ and $N_j$, belonging to two different constant UAGs. Whenever the right elements of these source nodes become active in the same phase (i.e. if they are bound to the same variable filler) the *to1* node becomes active and projects an inhibitory signal to stop binding propagation from the source nodes to the target argument nodes.

### 4.4 A Rule Encoding Example

To demonstrate the entire rule encoding procedure, let us reconsider the following example rule:

$$p(X, X, Y) \rightarrow q(X, Y)$$

Encoding the rule starts by constructing the two predicate assemblies corresponding the antecedent and the consequent of the rule as follows:

$$p{:}\{X_1([0],[0]), X_2([0],[0]), Y([0],[0])\},$$
$$q{:}\{X([0],[0]), Y([0],[0])\}.$$

Then the intermediate mechanism is built between them, which involves the following procedure to determine the required sub-mechanisms.

First of all, syntax of the rule is checked to obtain UAGs. From the example rule, we will get

$$S_a = \{G_X, G_Y\}, \text{ where } G_X = \{p{:}X_1, p{:}X_2\},\ G_Y = \{p{:}Y\},$$
$$S_c = \{G_X, G_Y\}, \text{ where } G_X = \{q{:}X\},\ G_Y = \{q{:}Y\}.$$

Secondly, the number of arguments of each UAG in $S_a$ is examined. The antecedent of the example rule has two UAGs: $G_X$ for the repeated variable arguments $X$'s and $G_Y$ for the single variable argument $Y$. Since $size(G_X) > 1$ and $size(G_Y) = 1$, only $G_X$ requires the binding collection sub-mechanism (BCM):

$$BCM = \{G_X\}.$$

Thirdly, the matching between UAGs in $S_a$ and those in $S_c$ must be checked. This is carried out by examining each UAG in $S_a$ which has the related UAG in $S_c$. Since both UAGs in $S_a$ have related UAGs in $S_c$, binding propagation sub-mechanisms (BPMs) are needed between the argument nodes of $G_X$ in $S_a$ and those of $G_X$ in $S_c$ as well as between the argument node of $G_Y$ in $S_a$ and that of $G_Y$ in $S_c$. This is indicated by

$$BPM = \{G_X, G_Y\}.$$

In the next stage, in order to determine the consistency checking sub-mechanisms, the type and size of UAGs in $S_a$ are checked. In the case of the example rule, we only need $CCM_v$ for the UAG, $G_X$, because it is a variable UAG and its size is greater than 1. This $CCM_v$ ensures the consistency condition where all variable fillers which are bound to these repeated arguments of $G_X$ should be the same constant filler or variable fillers. Thus we get

$$CCM_c = \{\},\ CCM_v = \{G_{X_s}\}.$$

Finally, binding interaction sub-mechanisms between UAGs will be accommodated. In the case of the example rule, a binding interaction sub-mechanism between variable arguments ($BIM_{vv}$) is needed for the pair of UAGs, $G_X$ and $G_Y$:

$$BIM_{cv} = \{\},\ BIM_{vv} = \{(G_{X_s}, G_{Y_s})\},\ BIM_{cc} = \{\}$$

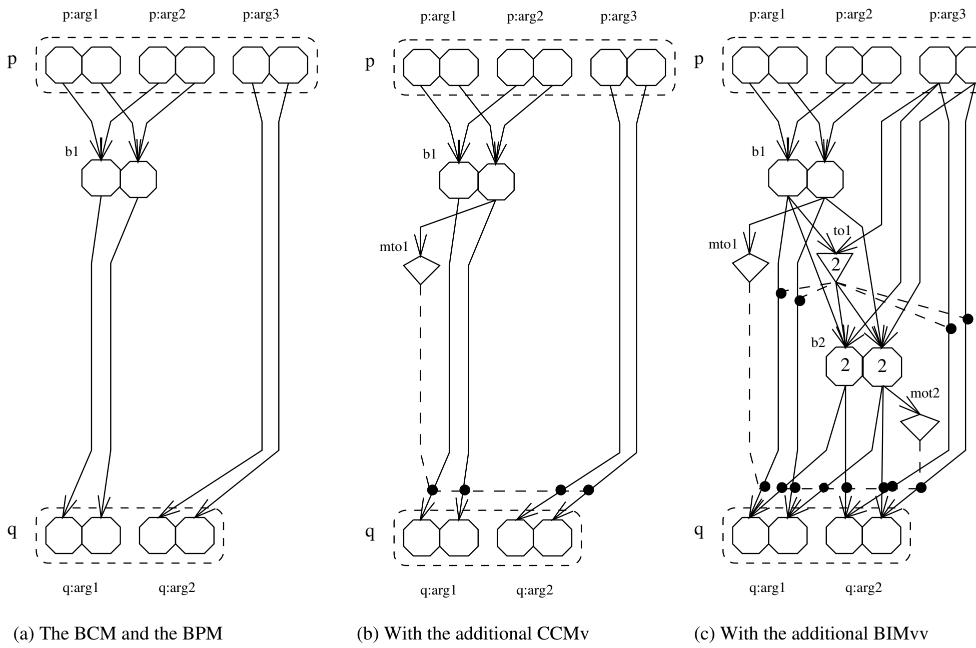


**Fig. 10** Building the intermediate mechanism for the rule $p(X, X, Y) \rightarrow q(X, Y)$

Fig. 10c shows the complete network generated for the example rule. Two base predicate assemblies are shown at the top and bottom of the network and between them are the $BCM$, the $BPM$, the $CCM_v$, and the $BIM_{vv}$. The binding node *b1* is used for the $BCM$, the *mto1* for the $CCM_v$, and the rest for the $BPM$ and the $BIM_{vv}$. Although the complete network looks very complex, it is modularised. Fig. 10a shows the network only with the $BCM$ and the $BPM$ and Fig. 10b with additional the $CCM_v$. As can be seen, the complexity of the network is mainly due to the binding interaction mechanisms between two UAGs. Rules which do not require this mechanism are encoded with a much simpler intermediate mechanism.

## 4.5 Performing Inference

Once a set of symbolic rules are encoded into corresponding SPNs, the inference architecture can perform forward chaining inference over the network. A step of inference is started by presenting a fact to the network. Presenting the fact involves specifying the argument bindings between the presented fact and the corresponding predicate assembly matched in the network. The SPN shown in Fig. 10c, for instance, presenting the predicate, $p(a,U,U)$, to the network will set up initial bindings, $\{a/p{:}X_1,\ U/p{:}X_2,\ U/p{:}Y\}$, by activating the filler nodes and their corresponding argument nodes in the following phases:

$$a[1],\ U[2],\ p{:}\{X_1[0,1],\ X_2[2,0],\ Y[2,0]\}$$

where the numbers specify particular phases. As inference continues, these activations propagate to the $q$ predicate assembly through the intermediate mechanism. This automatic binding propagation eventually activates the argument nodes of the $q$ predicate assembly as follows:

$$q{:}\{X[2,1],\ Y[2,1]\}.$$

The right elements of the argument nodes represent the constant bindings and the left elements the variable bindings. Consequently, the result of inference, $q(a,a)$, is obtained from the $q$ predicate assembly as well as the result of unification produced during the inference, {a/U}.

## 5 COMPARISON WITH OTHER WORK

Since the proposed inference architecture is inspired by Shastri and Ajjanagadde's connectionist system SHRUTI, it is interesting to compare their system with ours. Firstly SHRUTI's binding mechanism only represents constant bindings explicitly. It represents variable bindings by inactivity of argument nodes during inference. It does not, therefore, differentiate one variable binding from others which is a basic, and fundamentally important, feature of symbolic inference. Secondly, SHRUTI's rule encoding mechanism does not consider all the restrictions on variable binding that symbolic rules may impose. In particular, a sub-mechanism for binding interaction between groups of unifying arguments is not considered in their system at all. However our model can represent variable bindings because it uses an extended dynamic binding mechanism. Its rule encoding mechanism can encode rules which require binding interaction between groups of unifying arguments during inference.

Other two similar systems are described in Sun (1992) and Lange and Dyer (1989). Sun (1992) has proposed the network model, called CONSYDERR, for commonsense reasoning. It consists of two sub-systems one of which uses distributed representation, the other localist representation. The localist sub-system can deal with various knowledge representation issues such as consistency checking, unification, etc.. The major difference between his localist sub-system and our network model lies in the functional requirements of nodes in a network. We use only one type of assembly node which consists of entity nodes. The behaviour of the assembly is quite transparent because, apart from temporal behaviour, each element of the entity nodes in the assembly performs very simple processing (summing of weighted inputs and thresholding over temporal phases). Sun's sub-system, however, uses three different types of assemblies and unification, for example, is performed by what he calls a *complex assembly*. This complex assembly performs complex computations to support unification during inference (see p. 106 of Sun(1992)). Although it does not use temporal phases, the need for high computational ability of assemblies adopted and the use of abstraction which hides network details is beyond simple summation with thresholding and so demands much more sophisticated neural elements than we require.

Lange and Dyer (1989) describe a connectionist system, ROBIN, capable of performing high level inferences over structured connections of nodes that encode world knowledge in semantic networks. It uses structured connections of nodes to encode a semantic knowledge base of related frames. When constructing a structured semantic network, a special node called a *signature node* is attached to each node representing a concept. During inference, ROBIN permanently allocates a unique numerical signature to each constant and represents dynamic argument-constant bindings by propagating the signature of the appropriate constant to the argument to which it is bound. Therefore, it can maintain large number of dynamic bindings and deals with rules having multiple variables. A problem with the use of signatures is that the signatures become high precision quantities if each entity has a unique signature. Thus, propagating bindings will require nodes to propagate and compare high precision analog values. Another limitation is that their model does not deal with consistency checking and unification issues.

## 6 CONCLUSION

Any connectionist architecture which aspires to replicate symbolic inference should be able to represent dynamic bindings and rules with embedded control mechanisms. Shastri and Ajjanagadde's SHRUTI is a step in this direction but it does not provide a connectionist equivalent of a naming system for unbound variables, which is a fundamental requirement in all of the standard symbolic reasoning systems. In addition, it does not consider all the binding restriction specified in symbolic rules when it encodes them. This limits the expressive power of their model.

The inference architecture proposed in this chapter shows how oscillation of neurons can efficiently be used to support constant and variable bindings using entity nodes with simple behaviours. This new rule encoding mechanism provides a way of representing not only a group of unifying arguments but also many groups of such arguments with appropriate consistency checking between groups. This extends the expressive power of the proposed inference architecture. Because of the space limitation, we do not deal with encoding facts into the corresponding SPNs. Although described here is a forward chaining style inference, the basic concept of the proposed connectionist architecture can be used to encode facts to support backward chaining inference with only slight adjustments [Park and Robertson (1995)].

The proposed inference architecture provides a mechanism which translates a significant subset of first-order Horn clause expressions into a connectionist representation which may be executed very efficiently. However, in order to have the full expressive power of Horn clause FOPL, we need to add the ability to represent structured terms as arguments of a predicate and recursion in rules. Currently, no connectionist system has provided a convincing solution to all these problems and to do so, in general, we would have to sacrifice one of the most attractive features of this class of system: its ability to guarantee an answer in finite time.

## ACKNOWLEDGEMENTS

This work was partly supported by the Ministry of Science & Technology, Korea, and the British Council under grant SCOT/KOR/2923/37/A.

## REFERENCES

Ajjanagadde, V. & Shastri, L. (1991). Rules and variables in neural nets. *Neural Computation.* 3, 121–134.

Clossman, G. (1988). A model of categorisation and learning in a connectionist broadcast system. Ph.D. Dissertation, Department of Computer Science, Indiana University.

Eckhorn, R. (1993). Dynamic bindings by real neurons: Arguments from physiology, neural network models and information theory. Open peer commentary, *Behavioural and Brain Sciences.* 16(3), 457 – 458.

Engel, A. K., Koenig, P., Kreiter, A. K., Gray, C. M., & Singer, W. (1991). Temporal coding by coherent oscillations as a potential solution to the binding problem: physiological evidence. In H. G. Schuster & W. Singer (Eds.), *Nonlinear Dynamics and Neural Networks.* Weinheim.

Fahlman, S. E. (1979). *NETL: A System for Representing Real-world Knowledge.* MIT Press.

Fodor, J. A. & Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: a critical analysis. In S. Pinkerr & J. Mehler (Eds), *Connection and Symbols* (pp. 3-71). Cambridge, Mass., MIT Press.

Horn, D., Sagi, D., & Usher, M. (1991). Segmentation, binding, and illusory conjunctions. *Neural Computation.* 3(4), 510 – 525.

Hummel, J. E., & Biederman, I. (1992). Dynamic Binding in a neural network for shape recognition. *Psychological Review.* 99(3), 480 – 517.

Lange, T. E., & Dye, M. G. (1989). High–level inferencing in a connectionist network. *Connection Science.* 1(2), 181 – 271.

Luger, G. F. & Stubblefield, W. A. (1989). *Artificial Intelligence and The design of Expert Systems.* The Benjamin/Cummings Publishing Company, Inc..

McCarty, J. (1988). Epistemological challenges for connectionism. Open peer commentary to 'proper treatment of connectionism' by P. Smolensky. *Behavioural and Brain Sciences*, 11(1).

Park, N. S., Robertson, D., & Stenning, K. (1995). An extension of the temporal synchrony approach to dynamic variable binding in a connectionist inference system. *Knowledge-Based Systems.* 8(6).

Park, N. S., & Robertson, D. (1995). A localist network architecture for logical inference based on temporal synchrony approach to dynamic variable binding. In R. Sun and F. Alexandre (Eds), The Working Notes of the IJCAI-95 workshop on Connectionist-Symbolic Integration. 63 – 68.

Shastri, L., & Ajjanagadde, V. (1993). From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences.* 16(3), 417 – 451.

Strong, G. W., & Whitehead, B. A. (1989). A solution to the tag-assignment problem for neural nets. *Behavioural and Brain Sciences.* 12, 381 – 433.

Sun, R. (1992). On variable binding in connectionist networks. *Connection Science.* 4(2), 93–124.

Wang, D. L., & Terman, D. (1995) Locally excitatory globally inhibitory oscillator networks. *IEEE Transactions on Neural Networks*. 6.