An Extension of The Temporal Synchrony Solution to Dynamic Variable Bindings in a Connectionist System*

Nam Seog Park[†], Dave Robertson[†], Keith Stenning[‡]

[†]Dept. of Artificial Intelligence, University of Edinburgh, Scotland
namseog@aisb.ed.ac.uk, dr@aisb.ed.ac.uk

[‡]HCRC, University of Edinburgh, Scotland
keith@cogsci.ed.ac.uk

Abstract

A structured connectionist model using temporal synchrony has been proposed by Shastri & Ajjanagadde. This model has provided a mechanism which encodes rules and facts involving n-ary predicates and handles some types of dynamic variable binding. Their model, however, has problems in dealing with important knowledge representation issues such as binding generation, consistency checking, unification, etc. This paper presents a restructuring of the original model which overcomes many of its limitations while, at the same time, reducing the number of types of node required and maintaining the merits of the original model.

1 Introduction

The relationship between symbolism and connectionism has been one of the major issues in recent Artificial Intelligence research. An increasing number of researchers from each side have tried to adopt desirable characteristics of the other. These efforts have produced hybrid models of cognition. One of the main technical difficulties in building hybrid modes has been the well-known dynamic variable binding problem [Feldman (1982), Barnden(1984), Malsburg (1986), and Fodor & Pylyshyn (1988)]. Many hybrid models proposed partial solution to the dynamic variable binding problem – for example: Touretzky & Hinton (1988), Dolan & Smolensky (1989), Lange & Dyer (1989), Smolensky (1990), Barnden & Srinivas (1991), Sun (1992) – but all of these impose strong representational limitations and/or rely on neuronal elements of high complexity.

*A research paper of the Department of Artificial Intelligence numbered RP93-666.

Recently Shastri & Ajjanagadde (1991, 1993) proposed temporal synchrony solution to the dynamic variable binding. Based on this solution, they have built a partial connectionist model for reflexive reasoning which permits very fast inference. They described in detail a technique for using simple *neuron-like* elements to encode rules and facts involving n-ary predicates with variables. A key part of their mechanism to solve the dynamic variable binding problem lies in matching rhythmic patterns of activity, temporal synchrony, within the network. This model has been shown to have a number of limitations in dealing with issues which are important in enabling the model achieving adequate symbolic processing capabilities. These issues, categorised by Sun (1992), involve pseudo binding, binding generation, consistency checking, and unification, etc.

We describe in this paper a fundamental cause of the limitations and suggest a way of overcoming them by extending Shastri & Ajjanagadde's system. Section 2 briefly describes the temporal synchrony solution, proposed by Shastri & Ajjanagadde (1993), to the dynamic variable binding problem and Section 3 illustrates some limitations of their model. In Section 4, we describe a possible extension of the mechanism by redefining a property of a ρ-btu node. Some consequent changes in encoding of rules and facts are also described. Section 5 shows new way of encoding rules facts using a redefined ρ-btu node. Section 6 discusses the extended mechanism and summarises our results.

2 A temporal synchrony solution to the dynamic variable binding problem

2.1 The solution

Shastri & Ajjanagadde (1993) observed that the the existence of rhythmic activity, temporal synchronous activity, in the brain suggests one solution to the dynamic variable binding problem in connectionist systems. They adopt an oscillation cycle in which all nodes in a network change their status according to their inputs and functional abilities. The key to their temporal solution to the dynamic variable binding problem lies in a separation of the period of the oscillation cycle into several phases and the use of phase-sensitive nodes of four different types.

Figure 1 shows the temporal behaviours of all phase-sensitive nodes. The number of phases in a period of oscillation is assumed to be 6 in the figure. A τ-and node in the figure produces an oscillatory pulse whose period of oscillation and pulse width matches that of the input from the ith oscillation cycle after it receives an oscillatory input in the $(i-1)$th oscillation cycle. A ρ-btu node receives a periodic spike train in the $(i-1)$th cycle and produces a periodic spike train that is in phase with the driving input from ith cycle. The behaviour of the τ-or and *multiphase* τ-or node look alike in that they produce an oscillatory pulse train whose pulse width is comparable to the period of oscillation on receiving the periodic spike train, with the following small

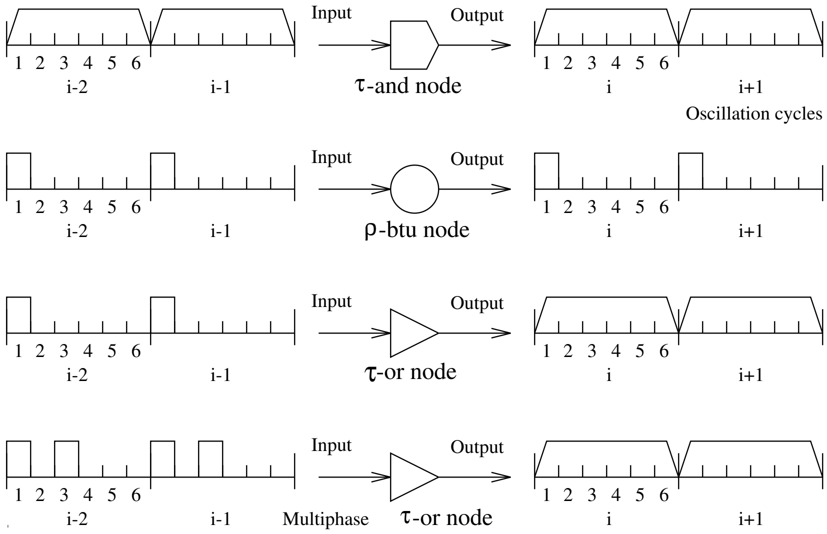


Figure 1: The temporal behaviour of phase-sensitive nodes

difference. A *multiphase* τ-or node becomes active only if it receives more than one spike trains in different phases within a period of oscillation as shown in the figure. On the other hand, τ-or node becomes active not only in the above situation but also when it receives one spike train from one of its input nodes within a period of oscillation as shown in the figure. The behaviours of these nodes are differentiated based on the window of time over which they sample their inputs and the width of their output pulse. The width of phase, ω, and the period of oscillation cycle, π, are limited to those indicated by physiological measurements.

2.2 A connectionist Representation of knowledge

Rules and facts which are described by the notion of the first order logic below are represented using phase-sensitive nodes and links as shown in Figure 2.

$$\forall X, Y, Z \;\; give(X, Y, Z) \rightarrow own(Y, Z)$$
$$\forall X, Y \;\; own(U, V) \rightarrow can\_sell(U, V)$$
$$give(john, mary, book1)$$

Each n-ary predicate in the figure is encoded using two τ-and nodes and n ρ-btu nodes where n is the number of arguments of the predicate. These two special τ-and nodes are referred to as the *collector* and the *enabler* of the predicate. Arguments contained in the n-ary predicate are encoded using n ρ-btu nodes. Each ρ-btu node corresponds to an argument in the predicate.

In the figure, the fact, $give(john, mary, book1)$, is encoded into a network using a τ-and node, called a *fact* node. This fact is associated with the *give* predicate

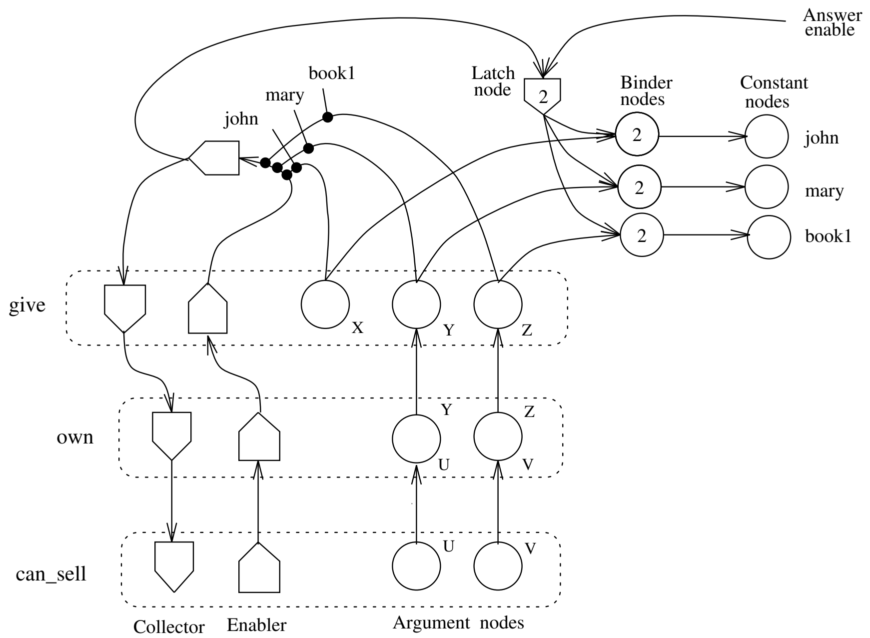


Figure 2: A connectionist representation of example rules and a fact

by which it is described. The fact node receives input from the *enabler* of the *give* predicate and this input is modified by inhibitory links from argument nodes of the predicate. The number of inhibitory links is equal to the number of individual constants in the fact.

Rules are encoded by linking matching arguments of the antecedent and those of the consequent. The direction of links of the network determines the type of inference a network can support. Backward chaining on this network is obtained by propagating variable bindings from the consequent to the antecedent in accordance with the argument correspondence specified in the rule. Forward chaining is done by propagating the argument variable bindings the opposite way. The network shown in the figure supports only backward chaining. The *latch* node and *binder* nodes in the figure are an auxiliary mechanism for answer extraction of the *wh*-type query processing. In order to support *wh*-type query processing, all facts in a network need this mechanism.

Once a network is established, the network is used to perform inference of two types, *yes-no* and *wh-type*. A query whose all arguments bound to constants, such as $own(mary, book1)$, corresponds to the *yes-no* question. However, a query which have existentially quantified variables can be interpreted as either *yes-no* or *wh-type* question. *own(mary,X)*, for example, can be interpreted as the *yes-no* question: "Can *own(mary,X)* be concluded from the rules and facts encoded in the reasoning system?" Or as the *wh-type* query: "For what values of X can *own(mary,X)* be concluded from the rules and facts in the system?" A type of query is determined by a user when one poses a query to the reasoning system. Posing queries to the network causes firing of initial nodes of the query predicate and inference on a network is achieved as a result of the propagation of the initial activation through the links from one predicate node to the other predicate node.

2.3 Representation of dynamic variable bindings

The temporal synchrony solution to dynamic variable bindings in the proposed encoding mechanism is obtained by argument nodes which share the same phase in any particular period of oscillation. A dynamic fact during a chain of inference is represented by a predicate for which each pair of argument and constant nodes is in phase. Figure 3 shows the dynamic representation of the fact, $own(mary, book1)$. The first argument of the *own* predicate and *mary* are sharing the second phase and the second argument and *book*1 are sharing the fourth phase. These in phase firings represent the variable bindings, $own\_arg1 = mary$ and $own\_arg2 = book1$.

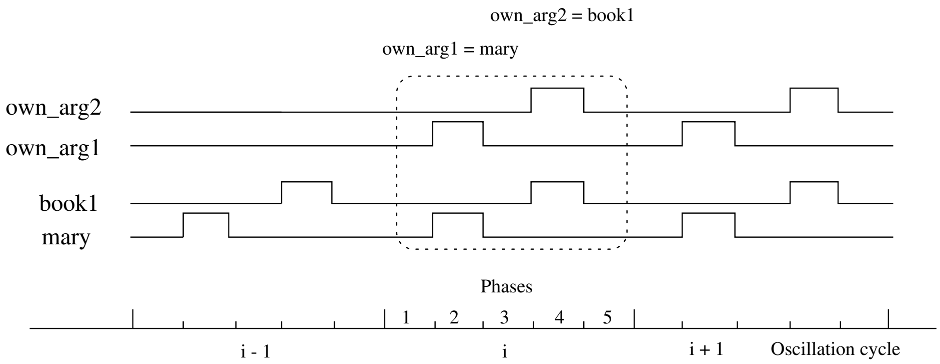


Figure 3: The representation of dynamic variable bindings

3 Exploring the model

As described in Shastri & Ajjanagadde (1993), their model can encode rules of the form:

$$\forall X_1, \ldots, X_k \;\; [p_1(\cdots) \wedge p_2(\cdots) \wedge \cdots \wedge p_n(\cdots) \rightarrow \exists Z_1, \ldots, Z_m \;\; q(\cdots)]$$

where the arguments of p_i's are elements of $\{X_1, X_2, \ldots, X_k\}$ and an argument of q is either an element of $\{X_1, \ldots, X_k\}$, or an element of $\{Z_1, \ldots, Z_m\}$.

Encoding these rules in a connectionist manner requires a network structure which is as uniform as possible and node functions which are very simple but at the same time are able to create dynamic symbolic structures. Shastri & Ajjanagadde's model firstly uses directed links to show the direction of inference between the antecedent and the consequent. Secondly, they use an assembly of nodes to represent a predicate and its arguments. One τ-and node and n ρ-btu nodes are required to encode an n-ary predicate. Finally, the relationship between the antecedent and the consequent is represented by setting up links between each node of the antecedent and a corresponding node in the consequent. To enforce special conditions of rules, an additional τ-or node and inhibitory links are used between them. This section describes how their encoding mechanism works and illustrates some limitations of their mechanism in dealing with further knowledge representation issues.

3.1 Dealing with a pseudo binding

The term "pseudo binding" refers to the situation where an unbound variable in a rule gets bound to another variable. Suppose we have the following rule and fact:

$$\forall X, Y \;\; own(X, Y) \rightarrow can\_sell(X, Y)$$

$$own(mary, book1)$$

Presenting the goal $can\_sell(mary, U)$ for backward chaining causes variable bindings between X and $mary$ and between Y and U. Here, since each argument node of *can_sell* may get bound to a different value, we need to differentiate one binding situation from the other in a connectionist manner.

Figure 4 depicts how the rule encoding mechanism in Shastri & Ajjanagadde's model deals with this issue. Figure 4a shows the way their system encodes the rule and fact and Figure 4b illustrates how bindings represented by phases propagate through the network. A pseudo binding in their mechanism is done simply by not providing any phases synchronised with a variable. If the query, $can\_sell(mary, U)$, is presented, the mechanism first activates the *enabler* of the *can_sell* predicate and allocates phases to constants and argument nodes. A constant binding between *mary* and the first argument of *can_sell* is obtained by allocating the same phase to both of them. In contrast, the mechanism does not allocate any phases to the variable U and to the second argument node of *can_sell* in representing a pseudo binding between them.

This pseudo binding as well as a constant binding is maintained through a chain of inference. Further applying the rule results in the activation of the *enabler* of the *own* predicate, the activation of the first argument of the *own* predicate in the same phase as *mary*'s, and again inactivation of the second argument node as a pseudo binding in the next cycle. This can also be seen in Figure 4. When the activation from the *enabler* of the *own* predicate reaches the fact node, there is no inhibitory signal coming either from the first or the second argument of the *own* predicate. So the fact node becomes active in the next cycle and this activation goes back to the *collector* of *can_sell* predicate eventually, which indicates a *yes* answer to the given

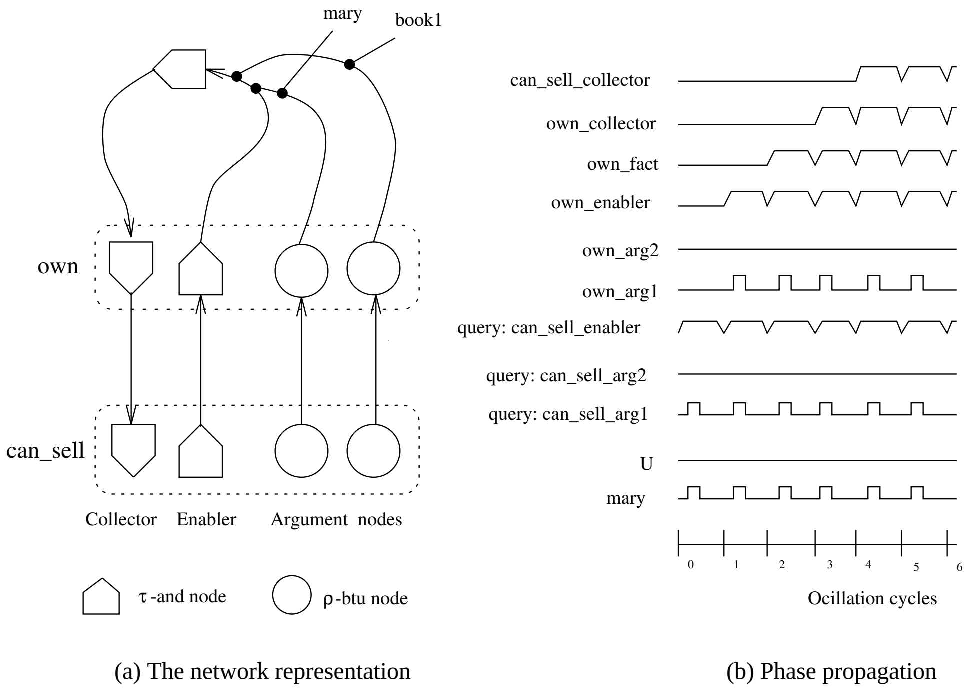


Figure 4: An example rule for a pseudo binding its propagation

query. In particular, when a query which contains all variables such as $can\_sell(X,Y)$ is posed to this mechanism, the system activates only the *enabler* of *can_sell* in the first oscillation cycle. This initial activation propagates to the *enabler* of *own* in the second cycle; then the fact node; the *collector* of *own*; and that of *can_sell* to succeed the query. In this case again, pseudo bindings of both variables, X and Y, are assumed simply by not allocating any phases to them.

As shown through the examples, this mechanism, as it stands, provides one way of distinguishing between different types of bindings, which gives the expected result when dealing with limited types of queries. However, when we consider the mechanism in relation to the more complex variable binding problems we discover that it is limited in its expressive power. We consider these limitations in sections 3.2 and 3.3.

3.2 Generation of bindings

The problem of binding generation can be illustrated using the following rule:

$$\forall X, Y \;\; p(X) \rightarrow q(X, Y, c1)$$

If a connectionist model performs forward chaining with this kind of rule (using the configuration in Figure 5a), generation of bindings is needed for the second and the third argument nodes of the q predicate since they do not receive any inputs from the antecedent. What is needed for the second argument is to generate a pseudo binding and for the third argument to generate a constant binding.

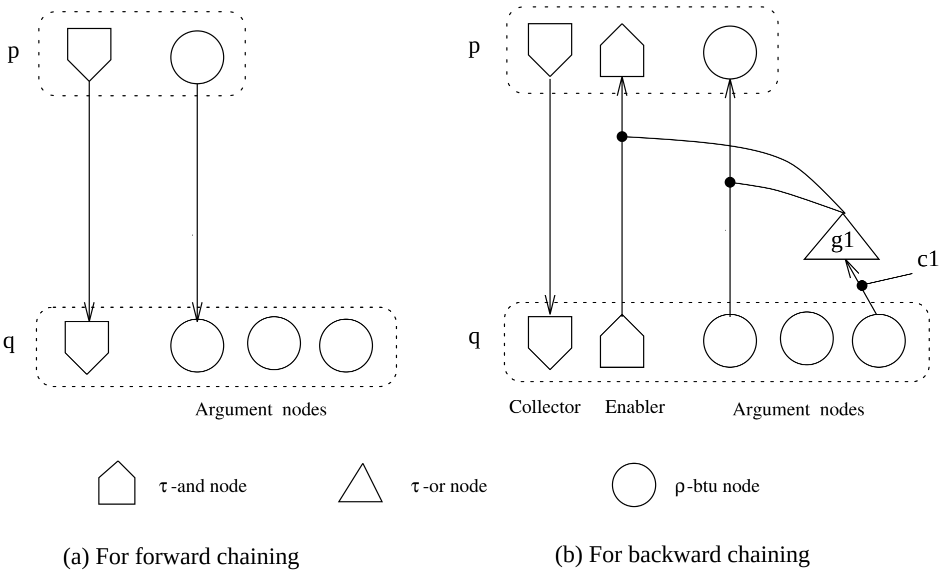


Figure 5: An example rule for bindings generation

On the other hand, if the model performs backward chaining on the same rule, the binding generation problem disappears (because Y is not used in satisfying $p(X)$) but we need consistency checking for the third argument of q since the rule should not be fired if that argument node is bound to a constant other than $c1$. Figure 5b illustrates this situation and the next section will discuss this issue in more detail.

In order to examine the binding generation issue more detail, we return to Figure 5a, which depicts possible way of encoding the given rule for forward chaining. Since Shastri & Ajjanagadde have not dealt with this issue in their model, their network does not have any additional mechanism for this purpose. Given $p(c2)$ to the rule, what we would like to conclude as a result of forward chaining is $q(c2, X, c1)$. To achieve this, the network should be capable of propagating the binding of the first argument of p to that of q. It also should generate a pseudo binding and a constant binding for the second and the third argument of q during the inference.

As can be seen in the figure, the propagation of a variable binding between the first arguments of p and q can be done in the usual manner through the link between them. Once the first argument of p gets bound to the constant $c2$ in the first cycle, this binding propagates to the first argument of q through the links between them in the next cycle. For the pseudo binding generation of the second argument node of q, the network naturally provides the effect of the binding generation due to the way they distinguish the pseudo binding from the constant binding. Generating a constant binding for the third argument of q, however, cannot be dealt with in their mechanism.

What is needed is an additional mechanism which provides the constant biding $c1$ to the third argument node of q during the inference. As long as this limitation is remained, the system cannot correctly infer the following type of rules: inferring $no\_of\_sides(a, 3)$, given $triangle(a)$ on the rule, $\forall triangle(X) \rightarrow no\_of\_sides(X, 3)$.

3.3 Consistency checking and unification

Consistency checking in a connectionist model requires us to keep track of bindings introduced by new constants and to keep track of repeated variables in predicate(s). The first situation can be illustrated by the rule shown in the previous section (see Figure 5b). When a query is posed for backward chaining, either a constant or a variable can get bound to the third argument of q. By the definition of the rule, if a constant, any_const, is assigned to the argument such as $q(X, Y, any\_const)$, consistency checking is necessary to make sure that any_const should be $c1$ in order to fire the rule properly. Consistency checking in Shastri & Ajjanagadde's model is carried out by a τ-or node, which is $g1$, and an inhibitory link from $c1$ as shown in figure. When a constant is assigned to the third argument of q by a query, if the constant is not $c1$, an inhibitory signal will be projected to the $g1$ node. Then the output from $g1$ will prevent activation flow from the consequent to the antecedent of the rule to prevent it from firing. So their mechanism provides consistency checking concerning a permanent constant binding. If the third argument of q gets bound to a variable, say U, when posing a query, the mechanism needs a way of unifying U and $c1$ to initiate inference. This type of unification is carried out in Shastri & Ajjanagadde's mechanism by the way they present the pseudo binding. Whenever the third argument of q unifys a variable during inference the argument node remains inactive and the $g1$ node does not project an inhibitory signal. Thus the rule fires normally.

The second type of consistency checking can be observed in the following rules:

$$\forall X, Y, Z \;\; p(X, Y) \wedge q(Y, Z) \rightarrow r(X, Z)$$

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

These rules have a repeated variable in the antecedent, which appears in the consequent as well. When the system performs forward chaining with these rules, the mechanism needs to force every occurrence of the repeated variable in the antecedent to get bound to the same value during inference. Thus consistency checking is required.

Figure 6 shows how these rules are encoded for forward chaining in Shastri & Ajjanagadde's model. To perform consistency checking, their encoding mechanism uses a τ-or node whose threshold is 2. In the first rule, the τ-or node becomes active when the variable Y gets bound to the same constant by asserting predicates such as $p(X, c1) \wedge q(c1, Z)$. Once the τ-or node becomes active, the activation reaches a τ-and node and a set of ρ-btu nodes. If all these nodes receive inputs from the

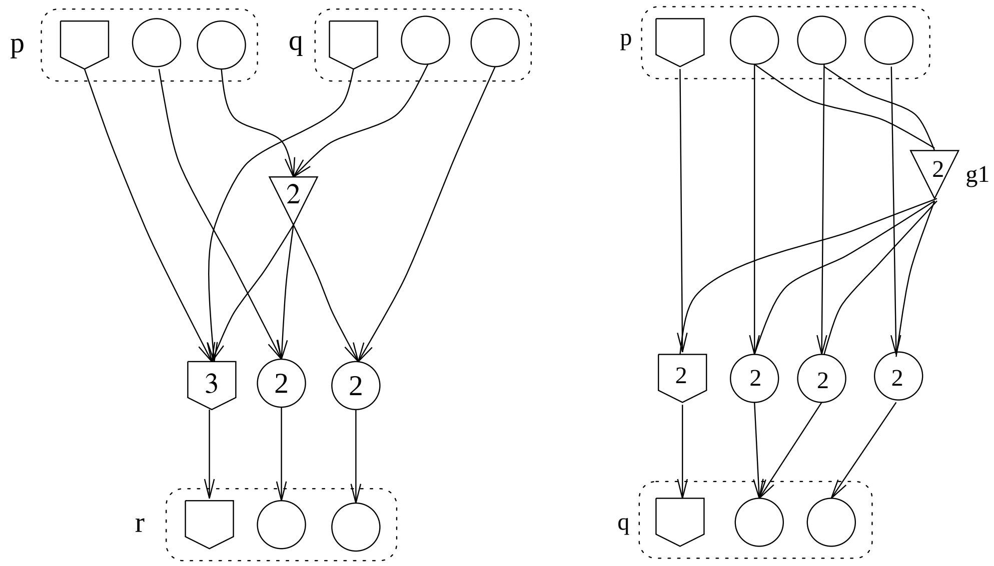


Figure 6: An example rules for a consistency checking and unification

antecedent, the rule fires. Consequently, the bindings propagate from the antecedent to the consequent.

In addition to consistency checking, unification is also required if repeated variable arguments get bound to variables. For example, when the predicates $p(c1, c2) \wedge q(X, c3)$ are presented to the antecedent of the first rule, the mechanism needs to unify X and $c2$. The mechanism, however, cannot properly deal with this unification because of the way it deals with a pseudo binding. Given the above query to the first rule, the second argument of p gets bound to $c2$ in a particular phase but a pseudo binding of the first argument of q is assumed without allocating any phases. As a result, the first argument of q does not project a phase output to the τ-or node whose threshold is 2. Inactive status of the τ-or node eventually fails in firing the rule. The same result can be obtained when the predicates, $p(c1, X, c2)$ or $p(X, Y, c1)$, are presented to the second rule. Hence, their mechanism has a significant limitation in dealing with unification which prevents it from handling transitive inference with variables although they provide special-purpose nodes for handling it. In section 4 we show how Shastri & Ajjanagadde's mechanism can be extended to cope with this problem with a more general, and simple, solution.

4 Extension of The Mechanism

From the symbolic point of view, a variable in a rule is considered something which can unify with a constant, a variable, or even a function. In a conventional symbolic system we have a free choice of algorithm for performing this task. In a connectionist system, however, we are more constrainted because of the uniformity of its structure and the simplicity of its processing elements. As discussed in the previous section, the temporal synchrony approach (although possessing the desired simplicity of structure) has some limitations in its representation of variable bindings. In this section we show how to overcome some of these limitations. Our extension hinges on the replacement of two of the four network node types (the ρ-btu node and multiphase τ-or) with a new type (the ω-btu node) which has a different behaviour.

4.1 Redefinition of a ρ-btu node

The limitation of Shastri & Ajjanagadde's model in dealing with unification is mainly due to their way of representing a pseudo binding. As long as a pseudo binding is represented simply by not providing any temporal phases, identifying bound variables is a problem.

One simple way of representing a pseudo binding may be to allocate an unused phase for the pseudo binding. This, however, makes it impossible to distinguish a constant binding from a pseudo binding because both of them will be represented by phases of the same type. An alternative way of representing it is by assigning continuous phases which make an argument node active at all phases within a whole cycle of oscillation. By this representation, the unification issue may be tackled with an additional mechanism but this requires modification of the temporal behaviour of a existing ρ-btu node.

We have adopted the second solution, and therefore need to define another node which has the required temporal behaviour and can be used as an argument node instead of a ρ-btu node. Figure 7 depicts the required behaviour of a new ρ-btu node which we call an ω-btu node. Binding of an ω-btu node to a constant is carried out in the same way as that of by the ρ-btu node. The only difference between the ρ-btu node and the ω-btu node is in a pseudo binding. As can be seen in the figure, the pseudo binding of the ω-btu node is now represented by an oscillatory pulse whose period of oscillation and pulse width matches the sum of all phases in a cycle. By this representation, the variable can unify with another value, a constant, or a variable, which are also represented by phases. In performing unification, the ω-btu node whose threshold is 2 becomes active on receiving two inputs within an oscillation cycle as follows:

- both of inputs are active at one of the phases. This is necessary when we have two constants inputs which we must ensure are the same.
- one of inputs is active at one phase and the other at all phases. This is the case when one input is a constant and the other a variable.
- both of inputs are active at all phase. This is the case when we have two variables inputs.

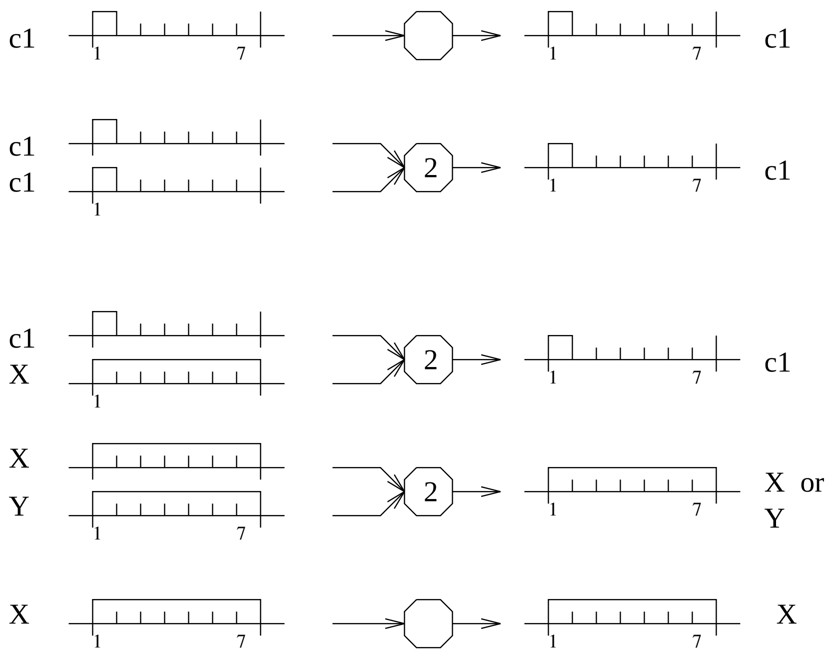


Figure 7: A new definition of a ρ-btu node (an ω-btu node)

4.2 Initialisation of queries

In Shastri & Ajjanagadde's mechanism, posing a query results in the following actions [Mani 1992]:

- activating constants: if the constant directly appears in the query, posing the query involves allocating a unique phase to that constant.

- activating the predicate(s): firstly, the enabler node for the query predicate is activated if the model performs backward chaining. Then the argument nodes of the query predicate which are bound to constants in the input query are activated such that an argument and its corresponding constant fire in synchrony, i.e. fire in the same phase.

- when the model performs forward chaining, there is slight difference in activating the predicate procedure but the basic concept is the same.

Since using an ω-btu node involves activating not only constants but also variables in a query, the following procedure is added to obtain the extended mechanism.

- activating variables: for a variable in the query, posing the query involves activating the variable in all phases in a cycle.

- activating the predicate(s): the argument nodes of the query predicate which are bound to variables in the input query are activated such that an argument and its corresponding variable fire in synchrony, i.e. both fire in all phases.

5 A knowledge representation using an ω-btu node

As mentioned in the previous section, adopting an ω-btu node requires some changes in encoding rules and facts. These changes, however, do not affect the two basic representations described in Section 3: the use of directed links to represent a direction of inference and the representation of each n-ary predicate using an assembly of nodes. The major change in encoding rules using an ω-btu node involves the introduction of connecting layers between the antecedent and the consequent of each rule to provide uniformity in constructing a network. This section describes in detail changes needed in these layers to deal with knowledge representation issues.

5.1 Ordinary rules and a pseudo binding

We first consider how rules similar to those handled by Shastri & Ajjanagadde's system may be represented. Figure 8b shows the new encoding of the following rule for backward chaining:

$$\forall X, Y \;\; own(X, Y) \rightarrow can\_sell(X, Y)$$

In the figure, the connection between the antecedent and the consequent are divided into two parts, the acknowledgement, and the gating. The acknowledgement layer consists of a τ-and node and a set of ω-btu and τ-or nodes. The number of ω-btu nodes and τ-or nodes in this layer is determined by that of different variables in the source predicates (i.e. the antecedent in forward chaining or the consequent in backward chaining). Each ω-btu node in the acknowledgement layer receives an input from a corresponding argument node and one of its outputs is sent to a τ-or node. The other output is connected to the corresponding ω-btu node in the gating layer. All outputs of the τ-or nodes in this layer are connected to a τ-and node as shown in the figure. Consistency checking, and unification are performed by the τ-and node through a set of ω-btu nodes and are acknowledged by a set of τ-or nodes and the τ-and node in this layer. The second layer, the gating, consists of a τ-and node and several ω-btu nodes. A number of ω-btu nodes is the same as that of argument nodes in a target predicate. The gating τ-and node receives input from the enabler of a source predicate, whereas a set of ω-btu nodes get their inputs from ω-btu nodes in the acknowledgement layer to propagate bindings. However, neither of them are fired unless they receive an input from the τ-and node in the acknowledgement layer. All the nodes in the gating layer can be shared among the rules which have the same target predicate. Binding generation is carried out in this layer. When the same rule

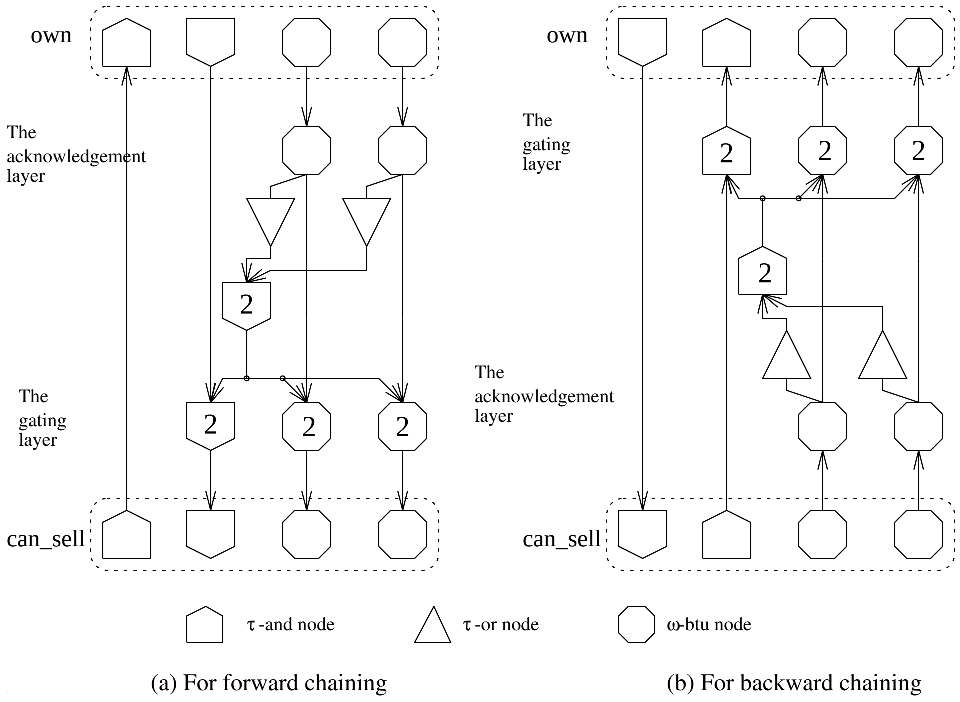


Figure 8: Encoding an ordinary rule using an ω-btu node

is encoded for forward chaining, the position of layers and the direction of all links should be opposite as shown in Figure 8a.

The dynamic variable binding and pseudo binding in this network is done quite simply. Suppose the query $can\_sell(mary, X)$ is posed to the network. The first thing that the extended mechanism carried out is as follows:

- allocating one phase, say in Phase 1, to the constant $mary$ and allocating all phases to the variable X for the pseudo binding, say in Phase *.
- activate the enabler of can_sell and argument nodes: the first one in Phase 1 and the second one in Phase *.

As the oscillation cycles goes on, these bindings of the can_sell predicate reach the gating nodes. At the same time an acknowledgement signal also arrive at the gating nodes. As a result, the gating nodes pass activation to the own predicate and a dynamic binding $own(mary, X)$ is obtained from the predicate, where the first argument of own oscillates in Phase 1 and the second argument of own oscillates in Phase *.

The layered connection mechanism in this example seems to be more complex than Shastri & Ajjanagadde's one. But, as we shall demonstrate, it scales up to accommodate more complex forms of inference in the same manner as ordinary rules do.

5.2 Rules which need binding generation

As discussed in Section 3.2, the binding generation is of two different types: pseudo binding and constant binding. The original model, as it stands, shows the effect of a pseudo binding generation by not providing any phases to corresponding argument nodes but any ways to generate a constant binding. To illustrate how these are dealt with in the extended mechanism, Figure 9 depicts a new way of encoding the example rule:

$$\forall X, Y \;\; p(X) \rightarrow q(X, Y, c1)$$

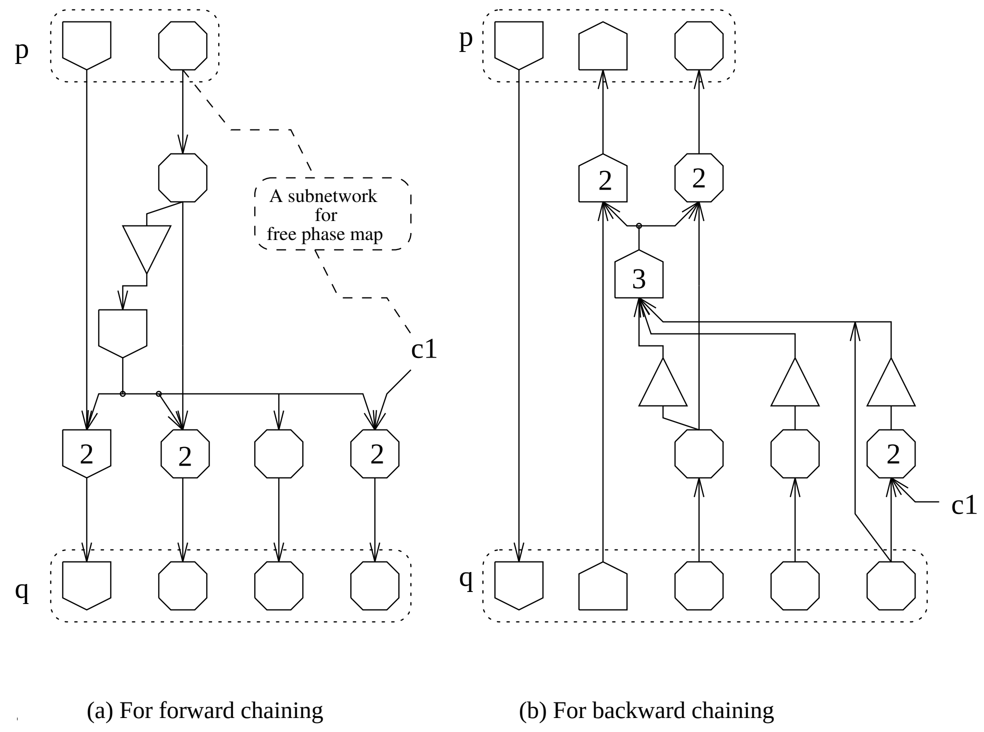


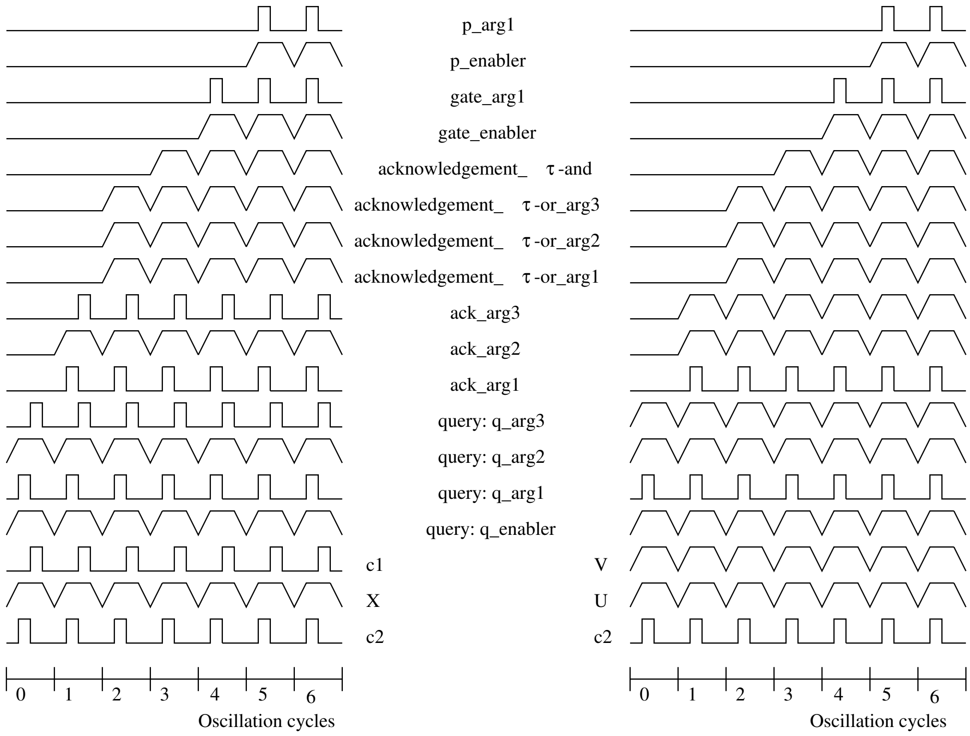
Figure 9: Encoding the rule which needs binding generation

Firstly, the representation of a pseudo binding for the second argument of q in the extended mechanism is carried out by interaction between the τ-and node in the acknowledgement layer and the corresponding ω-btu node in the gating layer. As can be seen from Figure 9a, the output of the τ-and node in the acknowledgement layer is connected to the ω-btu node for the second argument node of q. So whenever an acknowledgement is done by the acknowledgement layer, the output of the τ-and node generates a pseudo binding for the second argument of q.

Secondly, in order to generate a constant binding for the third argument of q, the third ω-btu node in the gating layer is dedicated for this purpose. This ω-btu node receives inputs from the τ-and node in the acknowledgement layer and from the constant node $c1$. The output of the node is connected to the third argument node of q as shown in the figure. During inference, once the constant node $c1$ is activated when the query such as $p(c2)$ is posed, the ω-btu node propagates the constant binding from $c1$ to the third argument node of q on receiving the other input from the τ-and node in the acknowledgement layer. As the result, $q(c2, X, c1)$ can be acquired from the predicate q. This additional mechanism, however, requires activating the constant node $c1$ in the middle of inference in other words, it requires automatic dynamic allocation of an unused phase to $c1$ after a query is posed. A subnetwork like the Free Phase Map found in Aaronson's (1991) Dynamic Fact Communication Mechanism (DFCM), can be used for this purpose. The function of the Free Phase Map in DFCM is to monitor the knowledge base and assigns, unused phases to other components of DFCM. Thus, if we adopt the subnetwork which performs the same task as the Free Phase Map, we can assign a unique, unused phase to the constant, $c1$, during inference and, in this way, the constant binding generation for the third argument of q can be achieved with easy. The dashed lines in Figure 9a give a brief idea how the Free Phase Map can be used in accordance with the main reasoning module.

5.3 Rules which need consistency checking and unification

Two types of consistency checking are required in encoding rules: single node consistency checking and multi-node consistency checking. The first type of consistency checking is required when backward chaining is performed on a rule which needs a constant binding generation in forward chaining, i.e. a rule which contains a constant in its conclusion (see Figure 9b). For a given query which has a constant in the third argument position, the network should perform consistency checking, whether the constant is $c1$ or not. As shown in the figure, this consistency checking is carried out by a corresponding ω-btu node, whose threshold is 2, in the acknowledgement layer. If the constant is $c1$, $c1$ and the third argument node of q fire in the same phase when posing the query. Consequently, the corresponding ω-btu node fires on receiving two the same inputs from them (see Figure 10a). However, if the constant is not $c1$, for example $c3$, the third argument node now will fire in synchrony with $c3$ and the constant $c1$ remains inactive. So the corresponding ω-btu node receives only one input from the third argument node and the rule does not fire. Thus consistency checking of the rule is maintained by the ω-btu node. For a given query which has a variable in the third argument position, unification should be performed by the network. Given $q(c2, U, V)$, for example, the network needs to unify the variable V with the constant $c1$. By posing the query, both V and the third argument node become active in all phases in the extended mechanism (see Figure 10b). when the activation reaches the corresponding ω-btu node, the node does not fire due to the inactive input coming from $c1$. So the activation goes through a by-pass link between the third argument of q and the τ-and node in the acknowledgement layer, thus unification of the rule is also maintained.



(a) Consistency checking with *q(c2,X,c1)* (b) Unification with *q(c2,U,V)*

Figure 10: Propagation of phases on the network shown in the Figure 9 when two different queries are posed to test consistency checking and unification

The multi-node of consistency checking and unification issues arises when rules have a repeated variable in the source predicate. The following rules illustrate this case:

$$\forall X, Y, Z \;\; p(X, Y) \wedge q(Y, Z) \rightarrow r(X, Z)$$

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

If both repeated variables get bound to constants, consistency checking between them is required. In the same way, if either one or both of repeated variables get bound to other variables, unification is necessary. In the extended mechanism, these are naturally carried out by the behaviour of the corresponding ω-btu node in the acknowledgement layer. As shown in Figure 11, the outputs of two argument nodes which should be bound to the same value are connected to the same ω-btu node whose threshold is 2. This node becomes active on receiving two the same constant inputs, or two variable inputs, or one variable and one constant inputs. The first rule, for example, fires when $p(X, c1) \wedge q(c1, Z)$ is presented since the argument nodes corresponding to the variable Y get bound to the same constant $c1$. The rule also fires when $p(X, Y) \wedge q(c1, Y)$ or $p(X, U) \wedge q(V, Y)$ are presented to the network. However,

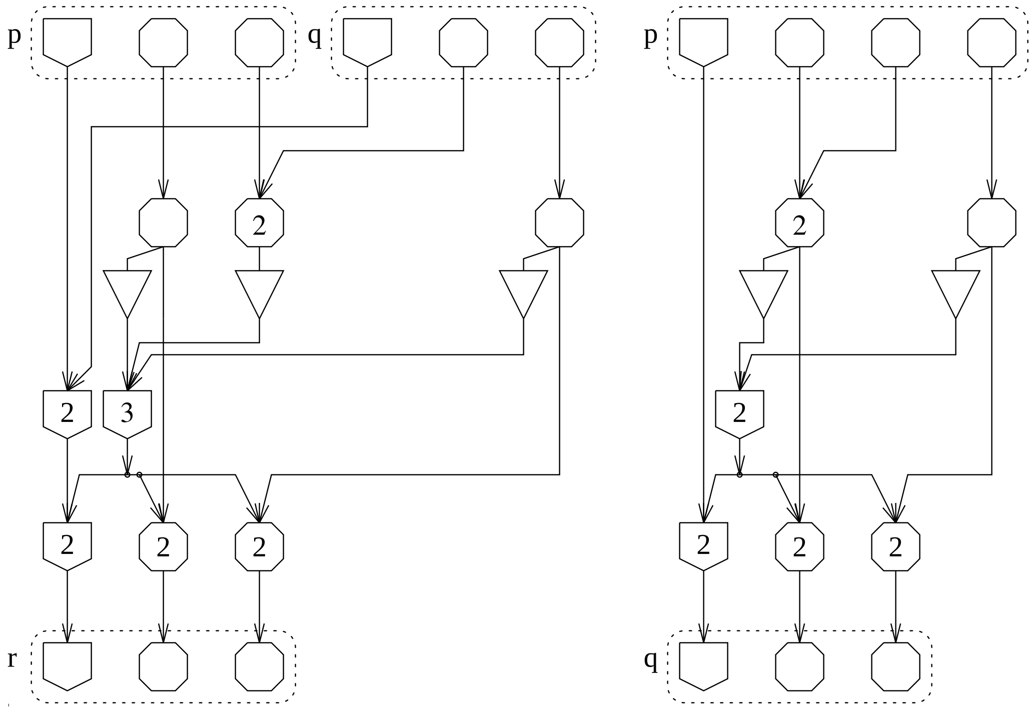


Figure 11: Encoding the rules which need consistency checking and unification

the rule does not fire if $p(X, c1) \wedge q(c2, Y)$ is presented. Similarly, in the second rule consistency checking and unification is carried out by the ω-btu node which receives inputs from two argument nodes which correspond to the repeated variable X.

5.4 Encoding of a fact

Unlike rules, facts are encoded by a constellation consisting of a τ-and node and the acknowledgement layer which consists of a τ-and node and a set of τ-or and ω-btu nodes as shown in Figure 12 for the fact $own(mary, book1)$.

The fact node has threshold 2 and receives inputs from the enabler of the predicate and the τ-and node in the acknowledge layer. Inhibitory links from the argument nodes to the fact node to encode a fact in Shastri & Ajjanagadde's model (see Figure 4) is replaced with the acknowledgement layer in the extended mechanism. Moreover, by this layer, inhibition to control the the activation of the fact node in the original model has turned to consistency checking by ω-btu nodes in the layer. To deal with a query which contains variables in argument positions, by-pass links are set up between all the argument nodes and the τ-and node in the layer. Since this fact encoding mechanism does not affect any change in associated mechanism for wh-type queries, the same wh-type query mechanism of original model can be used with the same concept.

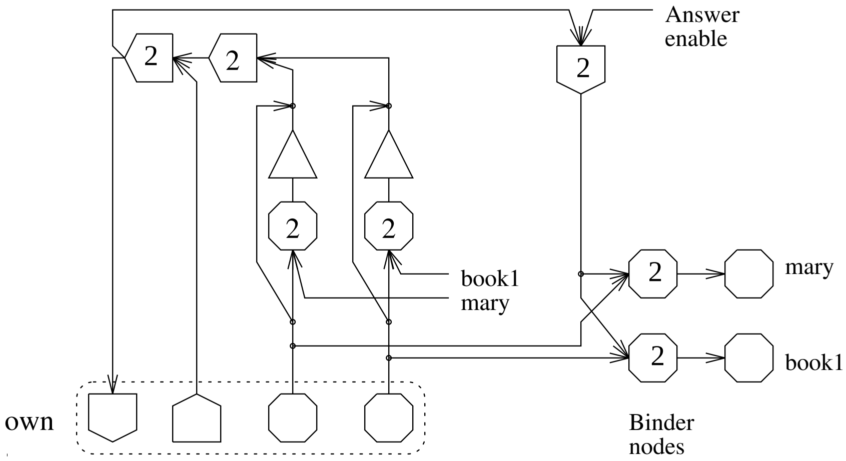


Figure 12: Encoding of the fact using an ω-btu node

6 Discussion

This paper has shown that how the temporal synchrony approach to the dynamic variable binding can be extended to deal with a broader range of knowledge representation issues based on first-order predicate calculus formulae than in Shastri & Ajjanagadde's original mechanism.

There have been some other approaches which have given partial solutions to these issues. Touretzky & Hinton (1985) can deal with binding generation automatically in a distributed fashion and also allows unification but only with very restricted forms of rules. Sun (1992) has suggested possible solutions to the issues. The major difference between Sun's approach and our extension of Shastri & Ajjanagadde's approach lies in the functional requirements of nodes in a network. In Shastri & Ajjanagadde's approach, only one type of assembly exists, constructured using four different node types. The behaviour of the assembly is quite transparent such that apart from their temporal properties each node in the assembly performs the same type of simple processing, weighted sums of inputs and thresholding. Sun's approach, however, uses three different types of assemblies. Ordinary assemblies compute weighted sums of inputs and pass along bindings for the variables; OR assemblies compute maximum of inputs and select one set of bindings out of many based on who is the winner; and Complex assemblies perform consistency checking, unification, and also binding generation when necessary. Although Sun's system does not use temporal phases, it imposes a higher computational requirement on each node, beyond simple summation with thresholding. For example, if Sun's system performs inference on rules where unification has to be done across predicates, for example:

$$\forall X, Y, Z \;\; p(X,Y) \wedge q(Y,Z) \rightarrow r(X)$$

Using Sun's system, the predicate node of the Complex assembly for r must perform the following computation in order to carry out unification properly.

> Suppose variables are represented by values greater than a certain threshold θ; if the value Y of the predicate p equals to the value Y of the predicate q, or if the value Y of p is greater than θ (which means variable), or if the value Y of the predicate q is greater than θ, then they are unifiable.

Although Sun's approach supplys one solution to the variable binding problem, the need for high computational ability in the assemblies and the use of abstraction which hides network details seems neurally implausible.

In our extension of Shastri & Ajjanagadde's system, a small change has been made in order to maintain transparent behaviour of the original system. Introducing an ω-btu node in encoding rules and facts has led us to adopt a layered connection mechanism between the antecedent and the consequent of rules. This gives the following advantages:

- All types of rules can be encoded only using τ-and, τ-or, and ω-btu. A *multiphase* τ-or node, used in Shastri & Ajjanagadde's original model, is no longer necessary.

- Inhibitory links to encode rules and facts in the original model are replaced by the acknowledgement layer which consists of ordinary links and nodes. This can make the implementation of the mechanism on hardware cheaper because links and nodes are cheaper than the control mechanism which is necessary for inhibition control.

- The translation from symbolic to connectionist representation is comparatively straightforward because of a uniform layered connection between the antecedent and the consequent of any types of rules.

Based on the representation of variable bindings using ω-btu nodes and the layered connection mechanism, the extended system has made it possible to tackle the knowledge representation issues more broadly while retaining simple network components and the neural plausibility of the original model.

References

[Aaronson 1991] J. Aaronson. Dynamic fact communication mechanism: A connectionist interface. In *Proceedings of the Thirteenth Conference of the Cognitive Science Society*, Lawrence Erlbaum.

[Barnden 1984] J. Barnden. On short-term information processing in connectionist theories. *Cognition and Brain Theory*, 7(1):25–59.

[Barnden & Srinivas 1991] J. Barnden and K. Srinivas. Encoding techniques for complex information structures in connectionist systems. *Connection Science*, 3(2):269–315.

[Dolan & Smolensky 1989] C. P. Dolan and P. Smolensky. Tensor product production system: A modular architecture and representation. *Connection Science*, 1.

[Feldman 1982] J. A. Feldman. Dynamic connections in neural networks. *Bio-Cybernetics*, 46:27–39.

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

[Lange & Dyer 1989] T. E. Lange and M. G. Dyer. High–level inferencing in a connectionist network. *Connection Science*, 1(2).

[Malsburg 1986] C. von der Malsburg. Am I thinking assemblies? In G. Palm & A. Aertsen (Eds), *Brain Theory*, Springer-Verlag.

[Mani 1992] D. R. Mani. Using the connectionist rule based reasoning system simulator. Technical Report 13, Computer and Information Science Department, University of Pensylvania.

[Shastri & Ajjanagadde 1991] L. Shastri and V. Ajjanagadde. Rules and variables in neural nets. *Neural Computation*, 3(1):121–134.

[Shastri & Ajjanagadde 1993] L. Shastri and V. Ajjanagadde. From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, 16(4).

[Smolensky 1990] P. Smolensky. Tensor product variable binding and the representation of symbolic structure in connectionist systems. *Artificial Intelligence*, 46(1):159–216.

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

[Touretzky & Hinton 1988] D. S. Touretzky and G. E. Hinton. A distributed connectionist production system. *Cognitive Science*, 12(3).