# An Extension of the Temporal Synchrony Approach To Dynamic Variable Binding in a Connectionist Inference 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**

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. A major open question in this field is the extent to which a connectionist architecture can accommodate basic concepts of symbolic inference, such as a dynamic variable binding mechanism and a rule and fact encoding mechanism involving n-ary predicates. One of the current leaders in this area is the connectionist rule-based system proposed by Shastri and Ajjanagadde. We demonstrate that the mechanism for variable binding which they advocate is fundamentally limited and show how a reinterpretation of the primitive components and corresponding modifications of their system can extend the range of inference which can be supported. Our extension hinges on the basic structural modification of the network components and further modifications of the rule and fact encoding mechanism. These modifications allow the extended model to have more expressive power in dealing with symbolic knowledge such as unification of terms across many groups of unifying arguments.

## 1 INTRODUCTION

At the intersection between symbolic inference and connectionism, there is interest in producing systems constructed from connectionist components which perform types of inference comparable to symbolic systems. The attraction of such systems is that, under restricted conditions, they may be capable of very fast, "reactive" responses to

*To appear in the special issue on "Knowledge-Based Neural Networks" planned by the journal *Knowledge-Based Systems*.

external stimuli. One of the main technical difficulties in building hybrid modes is the well-known *dynamic variable binding problem* [1-4]. Many hybrid models proposed partial solution to this problem – for example: DCPS [5], TPPS [6], ROBIN [7], COMPOSIT [8], CONSYDERR [9] – but all of these impose strong representational limitations and/or rely on neural elements of high complexity.

Subsequently, Shastri and Ajjanagadde [10,11] (henceforth S & A) proposed another form of dynamic variable binding using temporal synchrony of neurons. They described in detail a technique for using simple neuron-like elements to encode rules and facts involving n-ary predicates with variables and built a connectionist model for reflexive reasoning (called SHRUTI) which performs an interesting class of inferences extremely quickly. We consider the idea of using synchronous activity of neurons to represent bindings to be both efficient and elegant. However, S & A's mechanisms are subject to a number of significant limitations.

This paper diagnoses the shortcomings of SHRUTI and proposes a possible way of overcoming several of them. We first briefly review the main underlying idea of temporal synchrony approach, architectural components and terminology. Then we explore its main weaknesses. In the fourth section we describe in detail an extension needed to overcome the limitations of the temporal synchrony approach. The following section shows further modifications needed for the rule and fact encoding mechanism to be able to deal with some example rules and facts where the original system fails. Finally we introduce some related work and discuss our results.

# 2 THE TEMPORAL SYNCHRONY SOLUTION

S & A's system was inspired by the existence of rhythmic, temporal synchronous activity, in the brain. The key to their temporal solution to the dynamic variable binding problem lies in the propagation of phased oscillations through a network, consisting of phase-sensitive, neuron-like elements of four different types.

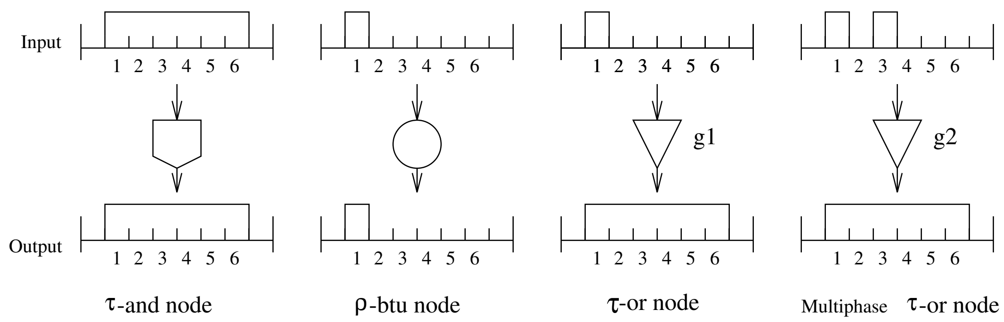



Figure 1: The temporal behaviour of phase-sensitive nodes

Figure 1 depicts the temporal behaviour of these nodes. The number of phases in one oscillation cycle is (for the purpose of explanation) assumed to be 6. A $\tau$-and node in the figure produces an oscillatory pulse whose period of oscillation and pulse width matches a period of oscillation after it receives the same oscillatory input. A $\rho$-btu node receives a periodic spike train and produces a periodic spike train that is in-phase with the driving input. A *multiphase* $\tau$-or node becomes active only if it receives more than one spike trains in different phases within a period of oscillation. On the other hand, a $\tau$-or node becomes active when it receives one or more spike signal within a period of oscillation. Both of them produce an oscillatory pulse train whose pulse width is comparable to the period of oscillation when activated. 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, $\omega$, and the period of oscillation cycle, $\pi$, are limited to those indicated by physiological measurements.

Using these basic elements, SHRUTI can encode subset of rules and facts of first order Horn-clause form:

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

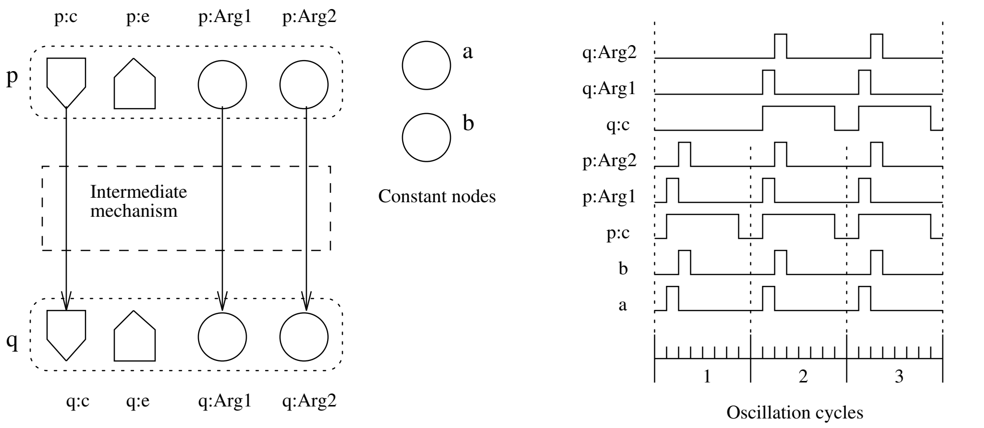
where $p_i$'s and $q$ are all positive atomic expressions. Expressions with no antecedent are normally called facts and expressions which have an antecedent and consequent are called rules.

Encoding each symbolic rule in SHRUTI consists of two procedures: building predicate assemblies corresponding to the antecedent and consequent predicates of a rule; and constructing the intermediate mechanism which connects the antecedent and consequent predicate assembly.

An $n$-ary predicate $p$ is encoded using two $\tau$-and nodes and $n$ $\rho$-btu nodes. These two special $\tau$-and nodes are referred to as the *collector* (*p:c*) and the *enabler* (*p:e*) of the predicate. The enabler of a predicate $p$ has to be activated whenever the system is queried about $p$. On the other hand, the system activates the collector whenever the dynamic bindings of the arguments of $p$ are consistent with the knowledge encoded in the system. The $n$ arguments of the predicate are then represented using $n$ $\rho$-btu nodes. For example, two predicates *p(X, Y)* and *q(X, Y)* in the rule $p(X, Y) \rightarrow q(X, Y)$ are represented as shown in Figure 2a.

In this figure, the intermediate mechanism is provided by a simple links from the antecedent to the consequent predicate assembly - thus propagating activation from one predicate assembly to the other. Note that these links are directed and this direction determines the type of inference a network can support: backward or forward chaining. Since the network in the figure has links from the antecedent predicate assembly to the consequent predicate assembly, it only support forward chaining. For backward chaining the direction of links should be reversed.

SHRUTI's fact encoding mechanism uses different sub-mechanism than that of rule encoding mechanism. Details of the fact encoding mechanism will be described in the later section, with an example.




(a) Predicate assemblies and constant nodes (b) Phasic representation of bindings

Figure 2: The representation of predicate assemblies and constant nodes

Once rules are encoded, a chain of inference on the network is carried out in the following way. When a fact which matches the antecedent predicate assembly is presented to the system, it activates the collector through the entire oscillation cycle. At the same time, it also activates fillers of the presented fact and the corresponding argument nodes of the predicate assembly in different phases to set up initial variable bindings between them. With the network in Figure 2a, presenting the fact *p(a,b)* makes the system activate the $a$ and *p:Arg1* nodes in the first phase and the $b$ and *p:Arg2* nodes in the second phase. These in-phase oscillations represent the variable bindings, $\{a/p{:}Arg1\}$ and $\{b/p{:}Arg2\}$. A variable binding between a particular argument node and a constant is obtained by making them share the same phase in any particular period of oscillation. The result of forward chaining on the network is obtained by propagating these initial bindings to the consequent predicate assembly through the intermediate mechanism (direct links in this case), which will eventually activate *q:Arg1* in the first phase and *q:Arg2* in the second. Figure 2b shows propagation of the initial bindings from the arguments of $p$ to those of $q$ in the form of phases.

S & A's system can produce some, but not all, the deductions we would expect from a Horn clause forward chaining system. In the next section, we describe some of the inferences which are not possible using S & A's mechanism. This provides the motivation for introducing our mechanism, which can handle these forms of inference correctly.

# 3 EXPLORING THE LIMITATIONS

Although S & A's system have demonstrated interesting class of inferences [11], we introduce in this section some standard inference patterns that their system cannot handle.

First of all, their system represents only a constant filler using a $\rho$-btu element and leave a variable filler unspecified. This prevents their system from performing unification across different groups of unifying arguments over encoded rules. Secondly, their fact encoding mechanism sometimes fails to force the constraints by an encoded fact when the fact contains a set of more than one occurrence of the same constant. Finally, when several facts which share the same predicate name and arity are encoded into a network, their system has difficulty in disambiguating the correct bindings between facts encoded and a query predicate posed.

We represent each problem situation using first-order Horn clauses as a standard language and show how S & A's system deals with these issues and where it fails.

## 3.1 Unification across different groups of unifying arguments

The first issue we will consider is how to encode a rule which requires unification across different arguments. Suppose the situation where the input fact $p(a,U)$ is presented to the following rule:

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

unification between $a$ and $U$ is required to force consistency among the repeated arguments of the rule. Firing of the rule should allow us to deduce $q(a)$ with unification result, $\{U/a\}$.

When the rule has additional arguments which have different names, we need to keep track of the unification result during inference, i.e. unification is required across unifying argument group. Suppose we add the additional argument $Y$ to the above rule:

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

then we will have two unifying argument groups for $X$ and $Y$. Presenting the input fact $p(a,U,U)$ now requires not only unification between $a$ and $U$ but also propagation of the unification result between two unifying argument groups. What is expected as the desired answer, in this case, is $q(a,a)$.

This type of unification is well supported by most symbolic rule-based systems using a symbolic matching procedure. However, connectionist systems are much more constrained in dealing with this type of unification because we require the network nodes to exhibit only simple, localised behaviour.

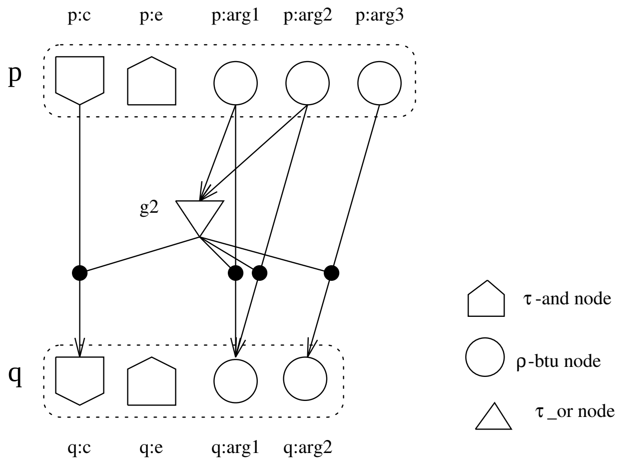



Figure 3: Encoding the rule which requires unification across different different groups of unifying arguments

Figure 3 depicts how the given rule is encoded using S & A's mechanism for forward chaining.

The predicates, $p$ and $q$, are represented using two predicate assemblies. The intermediate mechanism of the rule is then represented by connecting appropriate links between corresponding nodes of two assemblies. Since enablers are used only for backward chaining, there is no link between two enablers. As described in the previous section, presenting the fact to the system first allocates a unique phase to constant fillers and activates them and their corresponding argument nodes in the same phase, i.e. posing the query $p(a,U,U)$ results in:

phase allocation and constant activation:*a=[1], U=[$\phi$];*
predicate activation: *p:c=[*], p:e=[$\phi$], p:Arg1=[1], p:Arg2=[$\phi$], p:Arg3=[$\phi$]*

The symbol "*" stands for activation throughout the entire oscillation cycle, "$\phi$" represents the *null* phase[1] (no phase allocation), a number indicates the specific phase. This initial in-phase oscillation between $a$ and *p:Arg1* represents an initial variable binding, $\{a/p{:}Arg1\}$. Note that their system does not allocate any phases to variable fillers (*U's*) and their corresponding argument nodes (*p:Arg2* and *p:Arg3*).

Once initial variable bindings are set up, these bindings propagate to the consequent predicate assembly through the directed links. At this point, what we need is a way of forcing unification and the consistency condition on the first two arguments of $p$ because they are represented using the same variable name in the rule. Node *g2*, which is multiphase $\tau$-or node, is used for this purpose. As defined earlier, the g2 node only gets activated when two different constant fillers are assigned to *p:Arg1*

[1]To be precise, an inactive status of node throughout the entire oscillation cycle.

and *p:Arg2*. In case of the given input fact, *p(a,U,U)*, the g2 node will not become active because only the input coming from *p:Arg1* is active during inference. As a result, the *q* predicate will be activated in the following phases:

$$q{:}c=[*],\ q{:}Arg1=[1],\ q{:}Arg2=[\phi]$$

This indicates incorrectly that the answer *q(a,U)* is obtained from the *q* predicate assembly. The reason we do not obtain *q(a,a)* as expected is that the unification restrictions in the antecedent of the rule are not carried forward to the consequent. If we explore further, even when the fact *p(a,U,b)* is presented to the network, the binding $\{U/a\}$ is only observed in the procedure of a step of inference not from the arguments of *q*. In addition, if the fact *p(U,V,b)* which has variable fillers in the repeated argument positions is presented, there is no way to represent the binding $\{U/V\}$ in S & A's rule encoding mechanism.

## 3.2 Consistency checking over encoded facts

When the fact $p(a,a,b)$ is encoded into a connectionist inference system, the system should be able to answer two types of queries, *yes-no* and *wh-type*. For example, the yes-no query *p(a,a,b)*? can be posed to see whether or not any fact which matches with the given query is encoded into the network. Also the wh-type query *p(a,Y,Z)?* may be posed to obtain the constants which will get bound to $Y$ and $Z$ of the fact if there is any encoded fact matched.

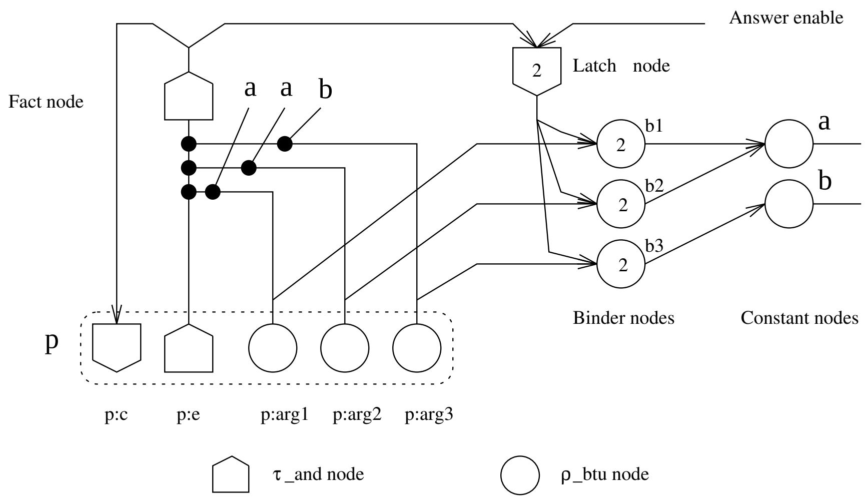



Figure 4: The network which encodes the fact *p(a,a,b)* using Shastri & Ajjanagadde's mechanism

SHRUTI's fact encoding mechanism uses a special $\tau$-and node, called a fact node, with inhibitory links to support yes-no query processing and an answer extraction mechanism for wh-type query processing. Figure 4 shows the fact node and the answer extraction mechanism for the encoded fact *p(a,a,b)*. As can be seen, the fact node has an input link from the enabler and an output link to the collector of the predicate. The input link from the enabler is modified by three inhibitory links from the argument nodes. These inhibitory links again are modified by the inputs from the constants to which the arguments of the encoded fact get bound. All these links and the fact node are used during yes-no query processing. To support wh-type query processing, the latch node and binder nodes are used as the answer extraction mechanism.

For better understand SHRUTI's fact encoding mechanism, let us think of the behaviour of the fact encoding mechanism when processing a wh-type query. Wh-type query processing requires two independent stages, firstly yes-no query processing and then answer extraction. When the wh-type query, *p(a,X,Y)?*, is given to the network shown in Figure 4.
Stage 1: firstly the system performs yes-no query processing to see if there exists a fact which matches with the given query in a network:

- posing the query *p(a,Y,Z)* to the system activates *p:e*. At the same time phase allocation occurs such that $a=[1]$, $Y=[\phi]$, $Z=[\phi]$. The corresponding argument nodes will also get activated in the same phases, $p{:}Arg1=[1]$, $p{:}Arg2=[\phi]$, $p{:}Arg3=[\phi]$. This establishes the initial dynamic bindings between them;

- in the next oscillation cycle, the active output from *p:e* activates the fact node since there is no inhibitory signal coming from the argument nodes. The output of *p:Arg1* is blocked by the same phase signal coming from the constant node *a*. The *p:Arg2, p:Arg3* do not project output signals because they are not activated during the query instantiation procedure;

- after next oscillation cycle the fact node activates the *p:c* this activation indicates an affirmative answer to the given *yes-no* query. The output from the fact node also activates the *latch* node which also receives input from the *answer enable*. This answer enable signal becomes active when a *wh-type* query is posed. Once a latch node becomes active it holds activation until the second stage of wh-type query processing finishes.

Stage 2: if the result of *yes-no* query processing is affirmative, the system must wait until all facts which match the query and their corresponding latch nodes in the network become active. Then the system performs the answer extraction procedure as follows. Otherwise inference stops and fails wh-type query processing.

- at the beginning of the second stage, the system now allocates distinct phases to the argument nodes corresponding to variable fillers of the input query which were left without allocated phases during the first stage. Consequently, new phase allocation is done for each argument node: $p{:}Arg2=[2]$, $p{:}Arg3=[3]$;

- this activation then propagates to the *binder nodes* which also receive active input from the latch node. Note that the threshold of binder nodes is 2. Nodes *b2* and *b3* become active in the first and second phase as a result;

- the activation of binder nodes then propagates to the constant nodes, resulting that *a=[1]* and *b=[2]*. The desired answer of the given wh-type query is obtained by observing in-phase activity between constant nodes and argument nodes, i.e. {*Y/a,Z/b*}.

Time taken to process each wh-type query is proportional to *2d*, *d* for the yes-no query processing stage and the other *d* for the answer extraction stage, where *d* is the maximum diameter of the network. If we explore this fact encoding mechanism for a series of other wh-type queries, we discover limitations in dealing with proper unification between the posed queries and encoded facts. As shown in Table 1, S & A's mechanism fails in forcing consistency on repeated variables in the input queries. Their mechanism can not produce proper answers to the the last three queries which require that any repeated variable in the input query should get bound to the same constant during inference.

| Query Posed | Desired Answer | S&A's Answer | Result |
|---|---|---|---|
| p(X,Y,Z) | Yes - {X/a,Y/a,Z/b} | Yes - {X/a,Y/a,Z/b} | right |
| p(X,X,Y) | Yes - {X/a,X/a,Y/b} | Yes - {X/a,X/a,Y/b} | right |
| p(X,X,X) | No | Yes - {X/a,X/a,X/b} | wrong |
| p(X,Y,X) | No | Yes - {X/a,Y/a,X/b} | wrong |
| p(a,X,X) | No | Yes - {a/a,X/a,X/b} | wrong |

Table 1: Response to the various queries presented to the encoded fact

## 3.3 Clarifying bindings among multiple encoded facts

A further limitation of SHRUTI's fact encoding mechanism is that it does not disambiguate bindings obtained by wh-type query processing over multiple facts (facts which share the same predicate name and arity). Suppose the following facts are encoded into a network:

$$f1 : p(a, b, c),\quad f2 : p(a, a, b),\quad f3 : p(a, a, a)$$

Since all facts have the same predicate name and arity, they will share the same predicate assembly *p*. Each fact requires its own answer extraction mechanism for wh-type query processing, which consists of a latch node and a set of binder nodes as described earlier.

If we pose the wh-type query, *p(X,Y,Z)* to the system, the system first performs yes-no query processing. During this stage, all three fact nodes and their corresponding latch nodes become active. At the beginning of the answer extraction stage, the system activates the argument nodes which correspond to the variable fillers:

$$p{:}Arg1{=}[1],\ p{:}Arg2{=}[2],\ p{:}Arg3{=}[3].$$

As inference cycles go on, the constant nodes, *a, b*, and *c*, will be activated by each set of binder nodes of the facts as follows:

by binder nodes of f1: $a{=}[1],\ b{=}[2],\ c{=}[3]$;
by binder nodes of f2: $a{=}[1],\ a{=}[2],\ b{=}[3]$;
by binder nodes of f3: $a{=}[1],\ a{=}[2],\ a{=}[3]$.

The final representation of these results will be:

$$a{=}[1;2;3],\ b{=}[2;3],\ c{=}[3].$$

where *a=[1;2;3]* represents that *a* node becomes active in the first, second, and third phases in one oscillation cycle. If a single $\rho$-btu node is used for each constant, these results cannot be represented properly because this type of node handles only single phase. S & A introduced a bank of predicates and a switching mechanism to address this problem. Instead of using a single node, rules and facts are encoded using a bank of $k$ predicates (see sections 5 and 6 their paper [11] for more details), where $k$ is the multiple instantiation constant.[2] Each constant node is also represented using a bank of $k$ $\rho$-btu nodes. If $k{=}3$, the above result, *a=[1;2;3]*, will be represented using the constant bank *a* and the rest of results using constant banks *b* and *c*.

From this representation, all that we can observe is that the constant *a* occupies the first, second, and third argument positions of the predicate *p*; the constant *b* the second and third argument positions; and the constant *c* the third argument position. These in-phase oscillations between constant banks and argument nodes which get bound to variable fillers do not necessarily disambiguate the desired bindings, $\{X/a,\ Y/b,\ Z/c\}$, $\{X/a,\ Y/a,\ Z/b\}$, $\{X/a,\ Y/a,\ Z/a\}$. The problem is that although we can tell which constants occupy which argument positions as the result of inference, we can not tell externally what the valid combinations are. Additional mechanism may be necessary to enable their fact encoding mechanism will be able to provide more information to disambiguate the corresponding bindings for each encoded fact.

Although using a bank of predicates and the switching mechanism provides a solution to multiple instantiation, enabling the system to support additional multiple instantiations greatly increase those space and time costs and structures, in terms of complexity and specificity of connections [12,13]. For example, in order to enable the system just to be able to support one more multiple instantiation, the system must replicate all predicates and constants once more as well as components of the switching mechanism.

[2]Because of significant space and time costs associated with multiple instantiation of predicates, $k$ has a fixed value 3 in S & A's system.

# 4 BASIC STRUCTURAL EXTENSIONS

In this section, we propose basic structural extensions of SHRUTI to overcome some of limitations we have described. The structural extensions involve modifying the structure of an argument node; the initial instantiation procedure; and the temporal behaviour of a $\rho$-btu node.

## 4.1 The structure of an argument node

In symbolic knowledge-based systems, each entity is represented using a unique symbolic name. This unique name is used to distinguish not only each entity from the others but also each binding from the others. A variable used in these systems can unify with a constant, a variable, or even a function in some inference systems but the system must always be able to identify which variable has been unified.

This observation suggests that equal treatment of constants and variables is necessary to extend the system. Since the temporal synchrony approach needs a distinct phase to tag each different entity, we have to assign a unique phase to each constant or variable node. However, assigning unique phases to variable and constant fillers and their corresponding argument nodes does not necessarily provide a way by which the extended system clearly separate the role of variables from that of constants during inference.

In order to differentiate the role of variables from that of constants in a network, we propose a slightly modified structure of the argument node. Instead of using a single $\rho$-btu node, we use a pair of $\rho$-btu nodes to represent each argument $Arg_i$: the left argument node ($Arg_{i_{left}}$) is used to represent a binding between a variable filler and the argument node (a pseudo binding) and the right argument node ($Arg_{i_{right}}$) to represent a binding between a constant filler and the argument node (a constant binding). In the diagrams which follows, we will use this convention to represent each type of a binding. Figure 5a depicts the modified structure of the predicate $p(X,Y)$. Note that we now use a dedicated node not only for each constant filler but also for each variable filler.

When the fact $p(U,a)$ is presented, the initial variable bindings between the fact and its corresponding predicate assembly, $\{U/p{:}Arg1, a/p{:}Arg2\}$, will then be represented by activating $U=[1]$, $p{:}Arg1=[1,\phi]$ and $a=[2]$, $p{:}Arg2=[\phi,2]$. The representation, $p{:}Arg1=[1,\phi]$, indicates that $p{:}Arg1_{left}$ is active in the first phase and $p{:}Arg1_{right}$ in the null phase. To be precise, the in-phase oscillation between $U$ and $p{:}Arg1_{left}$ is now representing the pseudo binding $\{U/Arg1\}$ and that of $a$ and $p{:}Arg2_{right}$ the constant binding $\{a/Arg2\}$. Therefore the types of bindings are distinguished by using different element of an argument node in the extended system. Figure 5b demonstrates the phasic representation of these bindings.

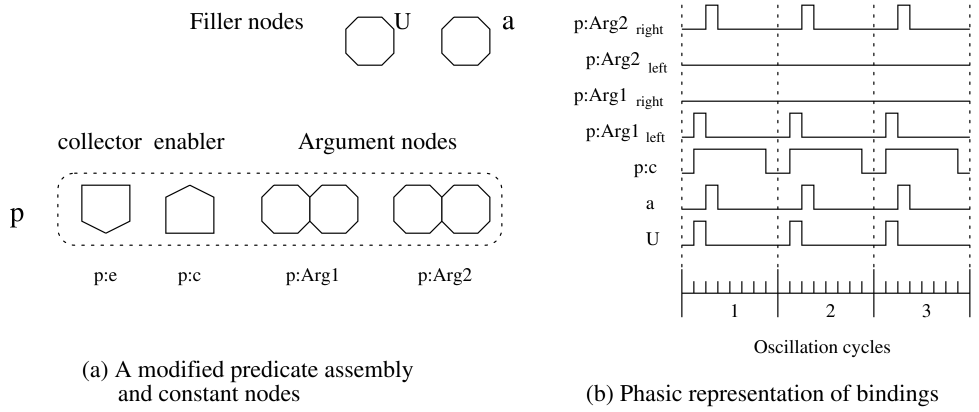



Figure 5: The modified structure of a predicate assembly

## 4.2 Initial instantiation of inference

Adopting a new predicate assembly with modified structure of argument nodes in encoding rules and fact requires some changes in the initial instantiation procedure. The extended system now needs the following procedure to establish initial variable bindings:

- activating fillers: presenting the fact involves allocating a unique phase to every filler (constants and variables) which directly appears in the input fact.
- activating predicates: firstly, the collector for the fact predicate is activated when the system performs forward chaining (or the enabler in the case of backward reasoning). Then each argument node gets activated in such a way that the system:
  - activates the right argument node and its corresponding constant in the same phase if the argument gets bound to a constant filler;
  - activates the left argument node and its corresponding variable in the same phase if the argument gets bound to a variable filler.

This procedure sets up initial variable bindings between the fillers of an input fact and the arguments of the corresponding predicate assembly. Once initial bindings are set up on the antecedent predicate assembly at the beginning of inference, further inference is carried out automatically by simply propagating these bindings to the consequent predicate through the intermediate mechanism between them.

## 4.3 Redefinition of a $\rho$-btu node

There is a tradeoff between a number of multiple instantiations allowed and the space and time complexity required in the system. As long as a $\rho$-btu node which accommodates only one input spike signal is used as an argument node, we are driven to using predicate and constant banks with a switching mechanism for multiple instantiations.

As an alternative solution, we propose the modification of the temporal behaviour of a $\rho$-btu node. The original temporal behaviour of the $\rho$-btu node is that when it receives a periodic spike train, it produces a periodic spike train that is in-phase with the driving input. We extend this temporal behaviour so that it can accommodate more than one spike signal in one oscillation cycle. A new temporal behaviour of a $\rho$-btu node (which we will call $\pi$-btu node) therefore will be described as follows:

> On receiving one or more spike signals in different phases, a $\pi$-btu node projects the same periodic spike pattern that is in-phase with the driving input. A threshold, $n$, associated with a $\pi$-btu node indicates that the node will become active in a phase of an oscillation cycle only when it receives $n$ or more spike inputs in that phase.

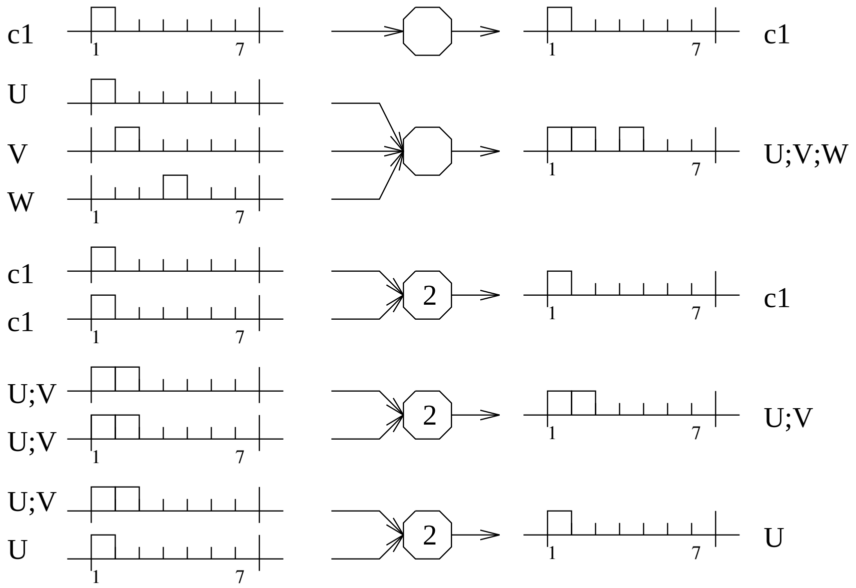



Figure 6: The temporal behaviour of a $\pi$-btu node

Figure 6 depicts a graphical demonstration of the temporal behaviour of a $\pi$-btu node (drawn as a hexagon). In each phase of an oscillation cycle, a $\pi$-btu node faithfully performs simple computations: summation of inputs and thresholding. In particular, when it has the threshold $n$, it passes through driving inputs only when it receives the same $n$ input spike signals in that phase. As a result of each computation in each phase, a $\pi$-btu node will show a temporal pattern over the full oscillation cycle and multiple instantiation of the $\pi$-btu node will be observed at this time scale. As shown in the figure, the notation $U;V$ will be used to represent that the $\pi$-btu node becomes active in both phases corresponding to $U$ and $V$ within one oscillation cycle.

In the next section, we will show that our system requires neurons to carry more information than in S & A's version but can represent multiple instantiations without predicate banks and the switching mechanism. Eckhorn [14] claims that the signal transfer rates on biological neurons are much higher than that used in S & A's mechanism. He adds that such high rates in single neurons are assumed to be capable of the signalling of complex messages, including the routing of dynamic representations. This means that there may exist a type of neuron whose signal transfer ability is even higher than our $\pi$-btu node.

# 5 EXTENSIONS OF ENCODING MECHANISMS

Since we have fundamentally altered the structure of S & A's original system, it is not surprising that we also need to change the way in which facts and rules are encoded. This section demonstrates these encoding, using the earlier examples where S & A's mechanism failed to produce the desired answers. To set the scene for this, we discuss some general properties of the encoding process.

## 5.1 Some considerations in encoding rules

When a symbolic rule is encoded, the antecedent and consequent predicates first are represented using modified predicate assemblies. Then the intermediate mechanism is constructed, which will connect the antecedent and consequent predicate. The complexity of the intermediate mechanism in encoding rules depends on the following factors:

- type of an argument: if a constant argument or repeated variable argument appears in the antecedent predicate, a consistency checking sub-mechanism is inserted into the intermediate mechanism. This sub-mechanism forces the condition that any constant filler which will get bound to a constant argument should be the same as the constant argument and any constant fillers which get bound to repeated argument should be the same constant;

- argument matching between the antecedent and consequent: if the same argument appears both in the antecedent and consequent predicates, a binding propagation sub-mechanism is required between those argument nodes to propagate an initial binding from the antecedent to the consequent predicate assembly. Those arguments which only appear in the antecedent do not need this sub-mechanism; however those arguments which only appear in the consequent require another sub-mechanism called the binding generation sub-mechanism;

- binding interaction between different groups of unifying arguments: when the same variable filler is assigned to numbers of different groups of unifying arguments, the appropriate bindings must be propagated to the other member of the groups.

In encoding each given symbolic rule, the extended system examines all the above factors first and then decide the structure of the intermediate mechanism. The intermediate mechanism will be more complicated when the antecedent predicate contains more than one group of unifying arguments and binding interactions are required between them.

## 5.2 The extended rule encoding mechanism

To exemplify how the extended rule encoding mechanism determines the necessary sub-mechanisms in encoding rules, let us return to the first of our earlier examples, which was the rule:

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

Encoding the rule starts with constructing the corresponding predicate assemblies of the predicates, *p(X,Y,Z)* and *q(X,Y)*. Then the intermediate mechanism between them is built, which involves the following procedure to determine the required sub-mechanisms.

First of all, a number of groups of unifying arguments is examined. There are two groups of unifying arguments - the $X$'s and the $Y$. The required sub-mechanism in this case is a consistency checking sub-mechanism for the repeated $X$'s in the precondition.

Secondly, the matching between arguments in the antecedent and those of in the consequent must be checked. The variable arguments $X$ and $Y$ appear both in the antecedent and consequent of the rule. Therefore binding propagation sub-mechanisms are needed between the first two arguments of $p$ and the first argument of $q$ and between the third argument of $p$ and the second argument of $q$.

Finally, binding interaction between different groups of unifying arguments will be accommodated. When the same variable filler is assigned to across different groups of unifying arguments, $X$'s and $Y$, the binding obtained from the argument group $X$'s must be propagated to the argument group $Y$. Therefore, we need bindings interaction sub-mechanism between these two groups of unifying arguments.

Figure 7 shows the network which encodes the given rule using the extended rule encoding mechanism. Two predicate assemblies are shown and between them is the intermediate mechanism. The *pb2* and *mto1* connections provide the consistency checking sub-mechanism for *p:Arg1* and *p:Arg2* ($X$'s in the given rule). The links from *pb1* and *pb2* to the *q:Arg1* and that from *p:Arg3* to *q:Arg2* are the binding propagation

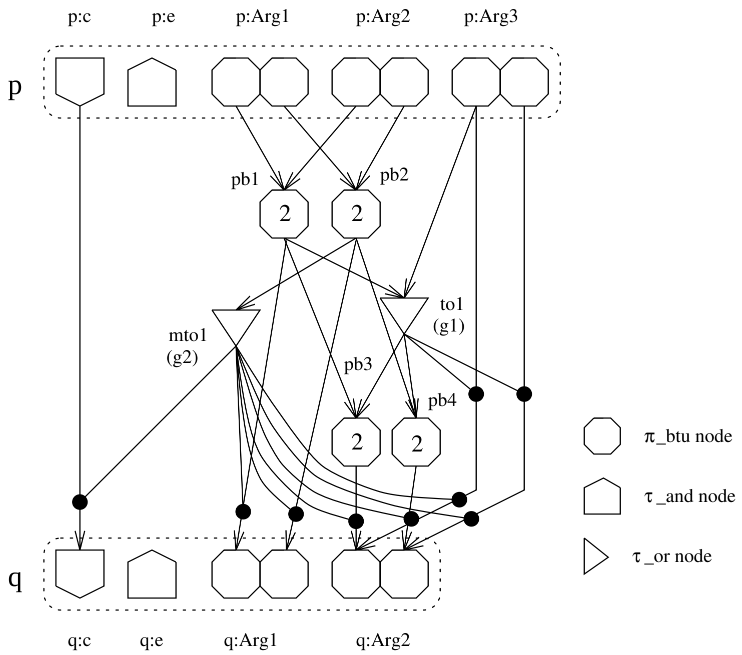



Figure 7: Encoding of the rule using the extended rule encoding mechanism

sub-mechanism for the arguments which correspond to $X$ and $Y$ in the given rule. Nodes *pb1*, *to1*, *pb3*, and *pb4* supply the binding interaction sub-mechanism between two different groups of unifying arguments $X$'s and $Y$.

When the fact *p(a,U,U)* is presented to the network, it should produce the desired answer, *q(a,b)*, with the unification result, $\{a/U\}$. Performing inference on the network starts with the initial instantiation of the $p$ predicate assembly which will result in the initial bindings:

allocation of phases: *a=[1], U=[2]*;
activation of the predicate: *p:c=[*], p:Arg1=[$\phi$,1], p:Arg2=[2,$\phi$], p:Arg3=[2,$\phi$]*.

This in-phase oscillation between the fillers and arguments represents the argument bindings, {*a/p:Arg1, U/p:Arg2, U/p:Arg3*}. Once the predicate is instantiated with these bindings, they are propagated to the consequent predicate assembly through the intermediate mechanism. In the second oscillation cycle, *pb1* becomes active in the second phase by the input coming from $p{:}Arg2_{left}$. While *pb1* in the first phase by the input from $p{:}Arg1_{right}$. These two nodes are used to accumulate inputs from the repeated arguments *p:Arg1* and *p:Arg2* which share the same argument name X in the given rule.

In the next cycle, *mto1*, which is a multiphase $\tau$-or node, receives input from *pb2* and checks if there are two different phase spikes in one oscillation cycle. This is to check consistency of the repeated arguments, i.e. if they get bound to two different constant fillers, this node will be active. Since *pb2* is only active in the first phase, *mto1* remains inactive. If *mto1* becomes active, its output will block the rule from firing. In the same cycle, *to1* will be active by two inputs from *pb1* and $p{:}Arg3_{left}$. This node checks if the same variable filler is assigned to different groups of unifying arguments. Because *p:Arg2* and *p:Arg3* get bound to the same variable filler U, this causes the activation of *to1* in this inference cycle.

As the inference cycle continues, the remaining $\pi$-btu nodes will become activated as follows:

*pb3=[2], pb4=[1]*

Finally, each argument node of the *q* predicate assembly will get active in the following phases:

*q:c=[*], q:Arg1=[2,1], q:Arg2=[2,1]*

This oscillation indicates the desired result of inference, *q(a,a)*, and also the result of unification, $\{U/a\}$.

As can be seen from the example rule, the extended rule encoding mechanism treats both pseudo bindings and real bindings equally. Unification across different groups of unifying arguments is handled by the binding interaction sub-mechanism during the binding propagation. The unification result is also easily obtained by observing each argument of the consequent predicate assembly in this extended rule encoding mechanism.

From this example, more complicated intermediate mechanisms employed by the extended system seems require more time for inference and a lager number of nodes in encoding rules than S & A's system. However, only rules which require a special treatment during inference such as unification across different groups of unifying arguments require such mechanisms in encoding rules. Rules without these features are represented with a very simple intermediate mechanism (only links) as shown in Figure 2.

The time taken to process a chain of inference in the extended system is proportional to $\bar{d}$, where $\bar{d}$ is the maximum diameter of a network which encodes the rules. The relationship between $\bar{d}$ and $d$ of S & A's system (without the multiple instantiation sub-mechanisms) will be $\bar{d} = \alpha d$, where $\alpha$ is the average ratio ($\bar{d}_i/d_i$) of extra steps of cycles required in encoding the $i$th rule of any type using the extended and S & A's rule encoding mechanism. In beast case, $\alpha = 1$ and $\bar{d} = d$ but in worst case, $\alpha = 2$ and $\bar{d}$ is two times longer than $d$. In general, $\alpha$ will have the value between 1 and 2 depending on the types of rules to be encoded. This may increase the diameter of the network and the number of nodes required. However, if S & A's system uses the switching mechanism and predicate banks for multiple instantiation, the $d$ should be multiplied by $k$, where $k$ is the multiple instantiation constant, and the total number of nodes required in encoding rules is even higher than that [11]. In this case, their system has much higher time and space complexity than those of the extended rule encoding system.

### 5.3 The extended fact encoding mechanism

The purpose of extending the fact encoding mechanism is three fold: to make it easy to represent the result of binding after query processing; to enable correct consistency checking over an encoded fact; and to reduce the time taken for wh-type query processing. The first extension is made by using a predicate assembly, called a fact assembly, to encode each fact rather than using a single fact node (a $\tau$-and node). The second extension is made by inserting a consistency checking mechanism between the fact assembly and the predicate assembly. The final extension is achieved by combining yes-no and wh-type query processing.

To explain each part of extension in more detail, let us return to our example fact $p(a,a,b)$. If we encode this fact using the extended fact encoding mechanism, the network is as shown in Figure 8. There are two intermediate sub-mechanisms in the network: the consistency checking sub-mechanism between the predicate and the fact assembly and the binding selection sub-mechanism between the fact assembly and the two constant nodes, $a$ and $b$. The consistency checking mechanism ensures constant and variable consistency during inference. Whenever a query is posed to the predicate, this mechanism examines if any constant filler of the given query gets bound to the one specified in the fact or if the same variable filler appear across different groups of unifying arguments. Node *pb1* and the fact inhibitory links are used for constant consistency checking and *pb2*, *pb3* and related links are used for variable consistency checking. When a posed query violates any forced consistency conditions, *to1* will be activated and this prevents the fact assembly from being active. Otherwise, the initial bindings will propagate to the fact assembly. In this case, the binding selection mechanism is used to activate the corresponding constant nodes of any variable fillers specified in the given query. This is done by propagating any pseudo binding of the fact assembly to the constant nodes through links. Once this is done, the binding selectors (*pb4* and *pb5*) are then used to select proper bindings for the right arguments of the fact assembly, which will eventually activate them. Inputs coming from the right argument nodes of the predicate assembly; activated constant nodes; and the left argument nodes of the fact assembly contribute to these binding selectors.

Let us reconsider wh-type query processing with the extended fact encoding mechanism. If we pose the query, $q(a,Y,Z)?$, to the extended system, posing the query will result activation of the predicate $p$ as follows:

$$\text{allocation phases: } a\text{=[1]},\ Y\text{=[2]},\ Z\text{=[3]};$$
$$\text{activation of the predicate: } p{:}e\text{=[*]},\ p{:}Arg1\text{=[}\phi\text{,1]},\ p{:}Arg2\text{=[2,}\phi\text{]},\ p{:}Arg3\text{=[3,}\phi\text{]}.$$

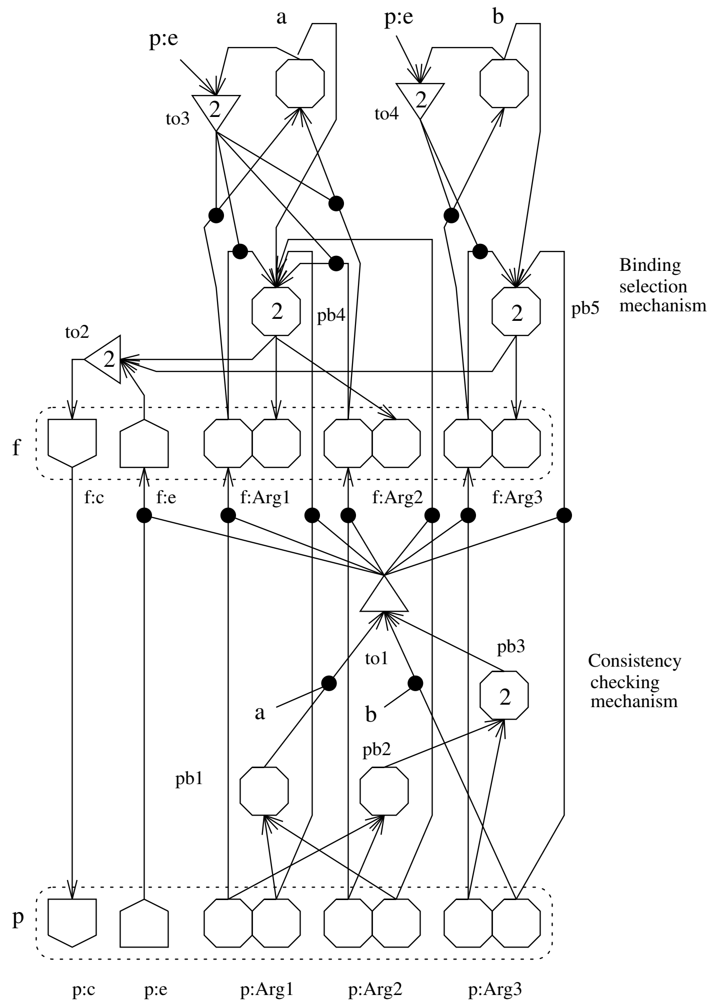



Figure 8: Encoding of the fact using the extended fact encoding mechanism

These initial bindings firstly propagate to the consistency checking mechanism. Because each filler of the query is either a constant or a variable, two sub-mechanisms are used to check both constant and variable consistency of the given query. In the next oscillation cycle, nodes *pb1* and *pb2* will be activated in the first and second phase. These nodes are used to accumulate the constant and variable phases of the repeated arguments, *p:Arg1* and *p:Arg2*, because both of them were represented using the same constant, $a$, in the given fact. The fact inhibitory link from *pb1* to *to1* and the other link between $p{:}Arg3_{right}$ and *to1* do not activate *to1* in the next oscillation cycle because the signal on the first link will be inhibited by the active input coming from $a$, while $p{:}Arg3_{right}$ remains inactive from the beginning of the inference. Similarly, the link from *pb2* to *pb3* and the other link from $p{:}Arg3_{left}$ to *pb3*, which are used for variable consistency checking, do not activate *pb3* because it has the threshold 2 but receives only one input signal from *pb2*. Consequently *to1* remain inactive with the given query and does not project an inhibitory signal. This means that the given query satisfies the consistency conditions of the encoded fact.

Once an initial binding passes through the consistency checking mechanism, the enabler and each left argument node of the fact assembly become active as follows:

$$f{:}e=[*],\ f{:}Arg1=[\phi,\phi],\ f{:}Arg2=[2,\phi],\ f{:}Arg3=[3,\phi]$$

These bindings then propagate to the binding selection sub-mechanism. This sub-mechanism first propagates all pseudo bindings appearing in the fact assembly to the corresponding constant nodes. The arguments *f:Arg2* and *f:Arg3* contains pseudo bindings these bindings will propagate to *a* and *b*. However, the binding propagation from $f{:}Arg2_{left}$ to *a* will be inhibited by the signal from *to3*. Node *to3* has been activated by the constant *a* and node *p:e*, which were activated during the initial binding instantiation procedure. As a result, *b* will be activated in the third phase. At this point, the binding selectors will select proper bindings for each right argument node of the fact assembly. Node *pb4* receives the same active input from *a* and $p{:}Arg1_{right}$ and becomes active in the first phase. On the other hand, *pb5* receives the same phase input from $f{:}Arg3_{left}$ and *b*, and becomes active in the third phase. When these active phases of the binding selectors propagate to the right arguments of the fact assembly, the final activation of the fact assembly will be

$$f{:}c=[*],\ f{:}a1=[\phi,1],\ f{:}a2=[2,1],\ f{:}a3=[3,3]$$

These represent the unification results, $\{Y/a, Z/b\}$, between the variable fillers of the given query and the arguments of the encoded fact to the given query, $p(a,Y,Z)$. The node *to2* is only used for timing to activate *f:c*. The activation of *f:c* will propagate to *p:c* and this indicates the success of query processing. By observing each argument node of the fact assembly, we can easily get not only the result of query processing but also details of the unification results, $\{Y/a, Z/b\}$, at the same time.

As shown in the above example, the extended fact encoding mechanism does not need different network configuration for yes-no and and wh-type query processing (as is the case in S & A's system). These two query procedures are merged into one fact encoding mechanism. This mechanism does not require two separate stages of wh-type query processing because the result of unification is automatically represented on the right argument nodes of the fact assembly during the query processing. Therefore, time taken to process each query is proportional to $\bar{d}$ not $2\bar{d}$ (as in S & A's system), where $\bar{d}$ is the maximum diameter of the network in the extended system.

## 5.4 Clarifying bindings among multiple encoded facts

Since the extended fact encoding mechanism provides uniformity in representing rules and facts, argument nodes of each fact can easily represent bindings related to them.

The extended system therefore can disambiguate bindings even when multiple facts are encoded under the same predicate name and arity at the same time.

Suppose we have encoded the following facts using the extended fact encoding mechanism:

$$f1 : p(a, b, c), \quad f2 : p(a, a, b), \quad f3 : p(a, a, a)$$

Posing the query *p(X,Y,Z)* to the extended system will first perform initial phase allocation, *X=[1], Y=[2], Z=[3]*, and activation of corresponding argument nodes of the *p* predicate assembly will result in the initial bindings {*X/p:Arg1, Y/p:Arg2, Z/p:Arg3*}. These bindings then propagate through each consistency checking sub-mechanism and binding selection sub-mechanism for each fact and eventually activate constant nodes and argument nodes of each fact assembly as follows:

a=[1;2;3], b=[2;3], c=[3]

*f1: Arg1=[1,1;2;3], Arg2=[2,2;3], Arg3=[3,3]*
*f2: Arg1=[1,1;2;3], Arg2=[2,1;2;3], Arg3=[3,2;3]*
*f3: Arg1=[1,1;2;3], Arg2=[2,1;2;3], Arg3=[3,1;2;3].*

The constant node *a* becomes active in *[1;2;3]* (the first, second, and third phases) in each oscillation cycle and the other constants *b* and *c* in *[2;3]* and *[3]*. Therefore by observing these temporal patterns of argument nodes of each fact assembly, we can obtain the answer of the given query. For example, from the first argument node of *f1*, *f1:Arg1=[1,1;2;3]*, the binding {*X/a*} is obtained and from the second and third argument nodes the bindings {*Y/b*} and {*Z/c*}. In a similar way, the bindings for the rest of the facts are obtained as follows:

*f2:* {*X/a, Y/a, Z/b*};
*f3:* {*X/a, Y/a, Z/a*}.

# 6 RELATED WORK

Several other connectionist systems which have offered solutions to the knowledge representation issues which we have discussed in this paper. Touretzky and Hinton's DCPS [5] can deal with unification but only with very restricted forms of rules.

Hölldobler and Kurfeß' CHCL [15] can handle unification over arbitrary first-order terms. However, the number of arguments allowed in each term is fairly limited because the total number of nodes required in their system is quadratic in the size of its knowledge base. Browne and Pilkington [16] have implemented CHCL using a distributed representation but their system still can not easily represent unification involving n-ary predicates.

Sun [9] has also suggested a possible solution to the unification problem. The major difference between Sun's approach and our extension of S & A's approach lies in the functional requirements of nodes in a network. S & A's system uses only one type of assembly node which consists of $\tau$-and nodes and $\rho$-btu nodes which were replaced by $\pi$-btu node in out extended system. The behaviour of the assembly is quite transparent since each node in the assembly performs the same type of simple processing (weighted sums of inputs and thresholding over temporal phases). Sun's approach, however, uses three different types of assemblies and unification is performed by what he calls a Complex assembly. This Complex assembly performs a high level of computation to support unification during inference (see p. 106 of [9]). Although Sun's system does not use temporal phases, the need for high computational ability of the assemblies and the use of abstraction which hides network details is beyond simple summation with thresholding.

In Park, et al [17] we gave a different, more limited, extension of S & A's mechanism for limited unification. We introduced a special temporal label, activated throughout an inference cycle, to tag variable fillers. We also used a double node to represent each argument node. A pseudo binding is then represented by clamping one of a pair of nodes to this temporal label and assigning an unused phase to the other as its tag. So all variable fillers and their corresponding arguments must propagate these temporal labels at each step of inference as well as interacting with other arguments. This extension, therefore, requires a more complex rule encoding mechanism than the one described in this paper.

# 7 DISCUSSION AND CONCLUSION

Any connectionist knowledge-based system which aspires to imitate symbolic inference should be able to represent unification across different groups of unifying arguments. S & A's system is fundamentally limited in this respect. As pointed out, the limitation of their system stems from the use of the same temporal phase (the null phase) to tag all variable fillers during inference. In our extension, both constant and variable fillers are represented using distinct phases. This equal representation has led to three major modifications of the original mechanism: the structure of an argument node, the replacement of a $\rho$-btu node by a $\pi$-btu node, and corresponding rule and fact encoding mechanisms. The use of a predicate assembly with modified structure of an argument node and the use of a new form of fact assembly in encoding rules and facts allows the extended system easily to represent the result of inference and unification on each argument node. The modified temporal behaviour of a $\pi$-btu node allows limited multiple instantiations of argument nodes without the switching mechanism and the use of a predicate bank. Finally, by the modification of rule and fact encoding mechanisms, the extended system provides a way of representing not only a group of unifying arguments but also many groups of such arguments even when it is not known to which individual those arguments are bound.

In addition to the unification issue, which is the topic of this paper, the extended system can also deal with other knowledge representation issues which are explored in Park, et al [18]. Entity categorisation using an ISA hierarchy and defeasible rules [11] in the original system can be also supported in our extended system without requiring special purpose additional mechanisms. Our extended system translates defeasible rules and ISA hierarchy definitions into the form of ordinary rules and facts and encodes them using the standard translation algorithm. This approach is in substantial agreement with that of Ajjanagadde [19].

The extension proposed in this paper provides a mechanism which translates a significant subset of Horn-clause into a connectionist representation which may be executed very efficiently (not only in forward chaining but also in a backward chaining style). However, in order to have the full expression power of Horn clause FOPL, we need to add the ability to represent recursion in rules and negation of predicates. Currently, no connectionist system has provided a convincing solution to implementing recursion in rules and a general solution would require us to sacrifice some desirable properties of our current system, in particular its guarantee of an answer to a query in finite time.

In spite of the introduction of a $\pi$-btu node, the extended system does not directly deal with symmetric or transitive rules which require multiple instantiations of the same predicate: such as $p(X,Y) \rightarrow p(Y,X)$ and $p(X,Y) \wedge p(Y,Z) \rightarrow p(X,Z)$. This is because all predicates which share the same predicate name and arity are represented using the same predicate assembly in the extended system. For example, when the fact *p(a,b)* is presented to the given symmetric rule $p(X,Y) \rightarrow p(Y,X)$, the system should be capable of representing both *p(a,b)* and *p(b,a)* on the predicate assembly *p(Arg1,Arg2)* at the same time in order to represent the result of symmetric inference. The cross connections between two arguments nodes can represent the bindings {*a;b/p:Arg1,a;b/p:Arg2*} but these do not tell us valid bindings {*a/p:Arg1,b/p:Arg2*} and {*b/p:Arg1,a/p:Arg2*}. To get round this problem, we have to represent the rule as follows: $p_1(X,Y) \rightarrow p_2(Y,X)$ where $p_1 = p_2$. This requires cooperation between the extended system and external equality checking mechanism.

One possible application of the extended system will be translating symbolic specifications to a sub-symbolic level. Although specifications expressed in a symbolic style are usually more convenient to read and manipulate, some domains may demand specifications at a sub-symbolic level (for faster run times or to enable inference using simpler components). This creates an extra layer in the software engineering process, with the symbolic specification acting as an intermediary between high-level requirements and sub-symbolic specification. The extended system proposed here could be used as an automatic translation mechanism from a symbolic specification language (a subset of Horn clause logic) to a sub-symbolic language (For more details see [20]). This could allow users to work at the symbolic level and have their descriptions "compile" to the sub-symbolic level.

## 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


1 Feldman, J A 'Dynamic connections in neural networks' *Bio-Cybernetics*. Vol 46 (1982) pp 27–39

2 Barnden, J 'On short-term information processing in connectionist theories' *Cognition and Brain Theory*. Vol 7 No 2 (1984) pp 25–59

3 von der Malsburg, C 'Am I thinking assemblies?' In Palm, G and Aertsen, A (Eds), *Brain Theory*, Springer-Verlag (1986)

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

5 Touretzky D S and Hinton, G E 'A distributed connectionist production system' *Cognitive Science*. Vol 12 No 3 (1988)

6 Dolan,C P and Smolensky, P 'Tensor product production system: A modular architecture and representation' *Connection Science*. Vol 1 (1989)

7 Lange, T E and Dye, M G 'High–level inferencing in a connectionist network' *Connection Science*. Vol 1 No 2 (1989)

8 Barnden, J and Srinivas K 'Encoding techniques for complex information structures in connectionist systems' *Connection Science*. Vol 3 No 2 (1991) pp 269–315

9 Sun, R 'On variable binding in connectionist networks' *Connection Science*. Vol 4 No 2 (1992) pp 93–124

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

11 Shastri, L and Ajjanagadde, V 'From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings using temporal synchrony' *Behavioral and Brain Sciences*. Vol 16 no 3 (1993)

12 Touretzky, D S and Fahlman, S E 'Should first-order logic be neurally plausible?' Open peer commentary, *Behavioural and Brain Sciences* Vol 16 No 3 (1993) pp 474 – 475

13 Palm, G 'Making reasoning more reasonable: Event-coherence and assemblies' Open peer commentary, *Behavioural and Brain Sciences.* Vol 16 No 3 (1993) p 470

14 Eckhorn, R 'Dynamic bindings by real neurons: Arguments from physiology, neural network models and information theory' Open peer commentary, *Behavioural and Brain Sciences.* Vol 16 No 3 (1993) pp 457 – 458

15 Hölldobler, S H and Kurfeß, F 'CHCL - A connectionist inference system' In Fronhöfer, B and Wrightson G (Eds) *Parallelization in Inference Systems*, Lecture Notes in Computer Science, Springer (1991) pp 318-342

16 Browne, A and Pilkington, J 'Variable binding in a neural network using a distributed representation' *ESANN94: Proceedings of European Symposium on Artificial Neural Networks*, Brussels (1994) pp 199 – 204

17 Park, N S, Robertson, D and Stenning, K 'Reasoning with limited unification in a connectionist rule-based system' Workshop on *Logic and Reasoning with Neural Networks*, 17 June, S. Margherita Ligure, Italy (1994)

18 Park, N S, Robertson, D and Stenning, K 'An extension of the temporal synchrony solution to dynamic variable bindings in a connectionist system' RP93-666, Department of Artificial Intelligence, The University of Edinburgh (1993)

19 Ajjanagadde, V 'Unclear distinctions lead to unnecessary shortcomings: Examining the rule vs fact, role vs filler, and type vs predicate distinctions from a connectionist representation and reasoning perspective', Proceedings of AAAI-94, Morgan Kaufmann (1994)

20 Robertson, D, Park, N S and Agusti, J 'Layered design of KBS from specification to hardware', ECAI-94 Workshop on *Formal Methods for Knowledge-Based Systems*, July, (1994)