A Localist Network Architecture for Logical Inference Based on Temporal Synchrony Approach to Dynamic Variable Binding

Nam Seog Park, Dave Robertson
namseog@aisb.ed.ac.uk, dr@aisb.ed.ac.uk
Dept. of Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
UK

Abstract

This paper describes a localist network architecture which translates a significant subset of Horn-clause logic into a connectionist representation which may be executed very efficiently. The proposed architecture is based on an extension of the temporal synchrony approach to dynamic variable binding originally proposed by Shastri & Ajjanagadde and provides a rule and fact encoding mechanism.

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. A major open question is the extent to which this type of connectionist architecture can accommodate basic concepts of symbolic inference.

Our primary interest is in understanding how systems which are described in a symbolic specification style may be implemented using a connectionist architecture. As a focus of attention we are developing techniques for "compiling" Horn clauses into a connectionist network. This offers significant practical benefits but also forces limitations on the scope of the compiled system. Executable symbolic specifications are (in the right hands) effective for describing systems and are comparatively easy for designers to understand. However, they normally require extra "machinery", in the form of an interpreter or theorem proving system, in order to be executed. For many applications - particularly when the system is to be implemented in hardware - such extra mechanisms are both inefficient and structurally complex. Connectionist system, on the other hand, use structurally simple components, may provide very fast inference and have no need of a separate interpreter, but are difficult to use directly for specification because of the mass of connections between elements. By providing automatic translation from symbolic to connectionist representations we should be able to cancel out the deficiencies of each style whilst retaining the advantages of both. Unfortunately, this type of compilation is not straightforward, since we are, in fact, merging an "interpreter" into the connectionist networks, the compilation process has to take into account not only the Horn clauses themselves but also the strategy which we intend to use for drawing inferences from them. It also appears that some fundamental aspects of symbolic inference are difficult to translate directly into a connectionist framework. In particular, it has proved difficult to provide a full translation of term unification (in the style of common Horn clause languages like Prolog) and this has, in turn, placed awkward limitations on the forms of inferences which could be supported. The purpose of this paper is to show how some of these fundamental limitation can be removed, resulting in a significantly extended system.

The proposed localist network architecture adopts an extension of the temporal synchrony approach proposed in Park, et al (1995) as the basic mechanism for dynamic variable bindings and provides a rule and fact compilation mechanism. This allows the proposed architecture can encode a significant subset of Horn-clause logic and extend the expressive power to deal with an important knowledge representation issues such as representing pseudo bindings, unification etc. [Sun (1992)].

The temporal synchrony approach has originally been proposed by Shastri & Ajjanagadde [Shastri & Ajjanagadde (1993), Ajjanagadde & Shastri (1991)] to address the dynamic variable binding problem in a connectionist system. They described in detail a technique for using simple neuron-like elements to encode rules and facts involving n-ary predicates with variables. On the basis of this mechanism, they built a connectionist model (called SHRUTI) for reflexive reasoning which performs an interesting class of inferences very quickly. However, Shastri & Ajjanagadde's mechanisms are subject to a number of limitations when compared to systems of symbolic inference such as Horn clause predicate logic [Park, et al (1995)].

Park, et al (1995) explored how a reinterpretation of the primitive neuron-like components and the rule and fact encoding mechanism of their system can extend the range of inference which can be supported. This extension provides the basic building blocks for the proposed localist connectionist architecture.

In the next section, we define some basic definitions which help to understand the rest of the paper. Section 3 shows how multiple variable bindings involving a single argument is represented using temporal synchrony. Sections 4 and 5 demonstrate rule and fact encoding mechanisms by showing the networks which encode a sample rule and fact. The next section explains how a step of inference is done over the network. Finally, Section 7 discusses the results and concludes the paper.

2 Basic Definitions

Definition: Our chosen subset of **first-order Horn clause calculus** is a set of universally quantified expressions in first-order predicate calculus of the 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. The conjunction of p_i is the antecedent and the q the consequent. An expression with no antecedent is called a *fact* and an expression which has both antecedent and consequent is called a *rule*. □

Definition A base predicate of an n-ary predicate, $p(A_1, A_2, \ldots, A_n)$, is a predicate which has the same predicate name p followed by the n argument names, $arg_1, arg_2, \ldots, arg_n$. All predicates which share the same predicate name and the same number of arity have the same base predicate. For each filler A_i of the given predicate, we say that arg_i of the base predicate is the corresponding argument of the filler A_i and all argument fillers must be either a constant or a variable. □

Definition: A **phase** is a minimum time interval in which a neuron-like element performs computation: sampling its input and thresholding. An **oscillation cycle** is a window of time in which neuron elements show their oscillatory temporal behaviours. Each oscillation cycle consists of several phases and the number of phases in an oscillation limits the number of distinct entities (constants and variables) which can be bound during inference. □

Definition: The **neuron-like elements** are the basic elements used to construct a network in the proposed inference system. Unlike ordinary nodes found in a conventional neuron model, these elements have a special temporal behaviour. The detailed temporal behaviour of each type of element is differentiated based on a type of input pattern they sample and a type of output pattern they produce. On becoming active, these elements continually produce the output (oscillating) for the duration of inference. Three types of neuron-like elements[1] are used in the proposed system:

- A **π-btu element** becomes active on receiving one or more spikes in different phases of any oscillation cycle. On becoming active, a π-btu element produces an oscillatory spike that is in-phase with the driving inputs.
- A **τ-or element** becomes active on receiving one or more spikes within a period of oscillation. On becoming active, a τ-or element produces an oscillatory pulse train whose pulse width is comparable to the period of an oscillation cycle.
- A **multiphase τ-or element** becomes active on receiving more than one inputs in different phases within a period of oscillation. On becoming active, a multiple τ-or element produces an oscillatory pulse train whose pulse width is comparable to the period of an oscillation cycle.

A threshold, n, associated with each neuron-like element indicates that the element will fire only if it receives n or more spike inputs in the same phase. □

Figure 1 shows graphic notation of each neuron-like element and its temporal characteristics.

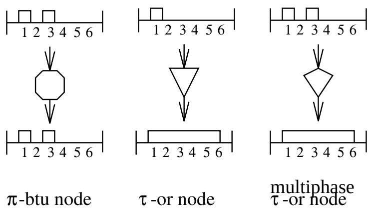

Figure 1: Temporal behaviours of the neuron-like elements

Definition: An **entity node** is a pair of π-btu elements. The left element is used to represent a variable role of the entity and the right element a constant role. A symbolic entity name may be used as a label of an entity node. Symbolically an entity node is represented using the notation, *entity_name*$[\phi,\phi]$, where the left position of the blanket represents the status of the left π-btu element and the right position that of the right π-btu element. The symbol "ϕ" denotes that the element is inactive. Also the subscripts, $_{left}$ or $_{right}$, are used as a suffix of a symbolic entity name to differentiate the left element from the right one. The notations, $entity\_name_{left}[\phi]$ or $entity\_name_{right}[\phi]$ are used to indicate each status of the left or right π-btu element. □

Definition: An n-ary **predicate assembly** is composed of n entity nodes, $arg_1, arg_2, \ldots, arg_n$. Each of these entity nodes is called an argument node. We use the notation, *pred_name:*$\{arg_1[\phi,\phi],\ arg_2[\phi,\phi],\ \ldots,\ arg_n[\phi,\phi]\}$ to represent an n-ary predicate assembly symbolically. When any individual argument node needs to be named, a prefix "*pred_name:*" is used before a symbolic name of the argument node. □

Figure 2 depicts how to represent a predicate assembly graphically.

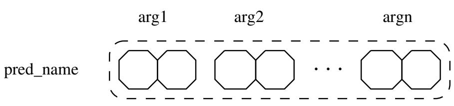

Figure 2: The structure of a predicate assembly

3 Representing multiple variable bindings

In the proposed system, each entity node is used as one of the following nodes: a constant node, a variable node,

[1] A τ-or and A multiphase τ-or elements are the same as defined in Shastri & Ajjanagadde (1993) and a π-btu element is newly defined.

and an argument node. When an entity node is used to represent a constant, only its right element becomes active during a chain of inference. Whereas when the entity node is used to represent variable, only its left node becomes active. However, when it is used to represent an argument of the predicate, either or both elements become active during a chain of inference.

For instance, the constant a, the variable U, and the argument arg_i are represented by entity nodes as follows:

$$U[Ph_1,\phi],a[\phi,Ph_2],arg_i[Ph_3,Ph_4]$$

where Ph_i is a particular phase in which the corresponding neuron element becomes active.

When constants, variables, and arguments of predicates involved in a set of symbolic rules and facts represented by entity nodes, they are used to represent variable bindings by making some of the neuron elements become active at the same phase of an oscillation cycle. If the right element the constant node a and that of the argument node arg become active in the same phase, this in-phase activation will represent a constant binding $\{a/arg\}$. In the same way a pseudo binding $\{U/arg_i\}$ can be represented by activating the left element of the variable node U and that of the argument node arg_i. If these two binding are represented together in the same oscillation cycle then multiple bindings, $\{a/arg_i\}$ and $\{U/arg_i\}$, involving the argument arg_i can be represented at the same time (this corresponds to binding a and U in a symbolic system). The proposed system uses this representation to distinguish between variable and constant binding in arguments - a feature absent in Shastri & Ajjanagadde's original system.

4 Encoding Rules

A symbolic rule in the from of the first order Horn clause calculus is encoded by building the antecedent and consequent predicate assemblies and the intermediate connectionist mechanism between them. These predicate assemblies are base predicates of the antecedent and consequent predicates of the symbolic rule. The intermediate mechanism connects the antecedent and consequent predicate assembly and provides a path for binding propagation. Consistency conditions of the rule are also forced by the intermediate mechanism. When being built, the complexity of the intermediate mechanism in encoding rules depends on the following factors of the given rule:

- 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 arguments should be the same constant;
- argument matching between the antecedent and consequent: if the same argument appears both in the antecedent and consequent, 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: if there are more than one different groups of unifying arguments in the antecedent of the rule, the appropriate bindings in one unifying argument group may need to be propagated to the other argument groups.

In encoding each given symbolic rule, the rule encoder must examine all the above factors first and then has to decide the structure of the intermediate mechanism required. Therefore the complexity of the intermediate mechanism of a rule depends on the complexity of the symbolic rule to be encoded.

To exemplify how the rule encoding mechanism determines the necessary sub-mechanisms in encoding rules, let us consider the following example rule:

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

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

First of all, a number of groups of unifying arguments is examined. The antecedent of the give rule has two groups of unifying arguments - the X's and the Y's. The required sub-mechanism in this case is a consistency checking sub-mechanism for the repeated X's in the antecedent so that any fillers which get bound to these repeated arguments should be the same constant or unifiable ones.

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 binding interaction sub-mechanisms between these two groups of unifying arguments.

The rule encoder automatically figures out the necessary sub-mechanisms for the intermediate mechanism and builds the corresponding network to the given rule.

Figure 3 shows the corresponding network which encodes the given rule. Two base predicate assemblies are shown at the top and bottom of the network and between them is the intermediate mechanism. The *pb2* and

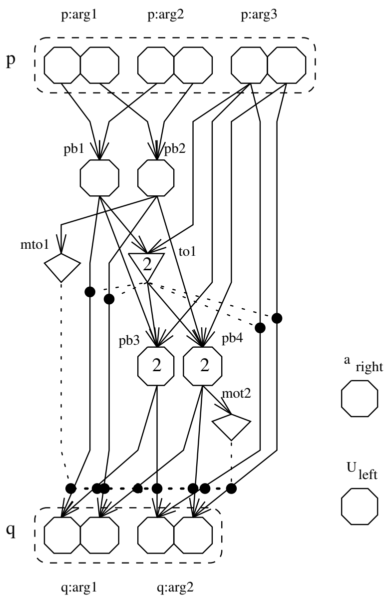

Figure 3: The network which encodes the rule $p(X, X, Y) \rightarrow q(X, Y)$

mto1 connections provide the consistency checking sub-mechanism for repeated arguments, *p:arg1* and *p:arg2* (corresponding X's in the given rule). The links from *pb1* and *pb2* to the *q:arg1* and those from *p:arg3* to *q:arg2* are the binding propagation sub-mechanism for the arguments which correspond to X and Y in the given rule. The nodes *pb1*, *to1*, *pb3*, *pb4*, and *mto2* are used as a binding interaction sub-mechanism between two different groups of unifying arguments X's and Y. The links drawn in a solid line are excitory and those in a dotted line are inhibitory. The eight black dots which are connected with a dotted line near to the bottom of the network are used to reduce the complexity of the network in drawing. On becoming active, any inhibitory sources connected to one of these black dots send the same inhibitory signal to the the rest of dots.

5 Encoding Facts

Facts are considered as a permanently instantiated predicate. The important sub-mechanism that has to be provided by the fact encoding mechanism is a means of forcing correct consistency checking over an encoded fact during query processing; and a sub-mechanism which makes it easy to represent the result of unification after query processing.

Since a fact can have more than one group of unifying arguments, some of which may be repeated arguments, two types of consistency need to be forced for each encoded fact. When posing a query, any fillers which get bound to arguments of the same unifying groups should be the same constant as specified in the arguments during query processing (constant consistency checking). In addition, when a variable filler appears across different groups of unifying arguments, the consistency checking sub-mechanism should detect this condition and prevent the fact being activated because this situation would bind the same variable filler to more than one constant argument at the same time (variable constancy checking). If a posed query satisfies all consistency checks, any variable filler of the query predicate needs to unify with its corresponding argument of the encoded fact. Thus, the fact encoding mechanism also has to represent results of unification between them.

Consider the encoded fact, *p(a,a,b)*, for instance. Posing the query, *p(a,a,b)* requires constant consistency checking; posing the query *p(a,X,X)* variable consistency checking; and for the query *p(X,X,Y)* unification between the variable filler X and Y and their corresponding constant arguments a and b.

To encode each fact, the fact encoding mechanism requires two predicate assemblies, a base predicate assembly and a fact assembly. The fact assembly has the same structure as an ordinary predicate assembly and is used to represent the result of inference involving the fact. The base predicate assembly provides the basic structure of facts to be encoded. All facts which share the same predicate name and the same arity are associated with the same base predicate assembly. This allows parallelism in inference over encoded facts. The consistency checking mechanism is built between the base predicate assembly and the fact assembly.

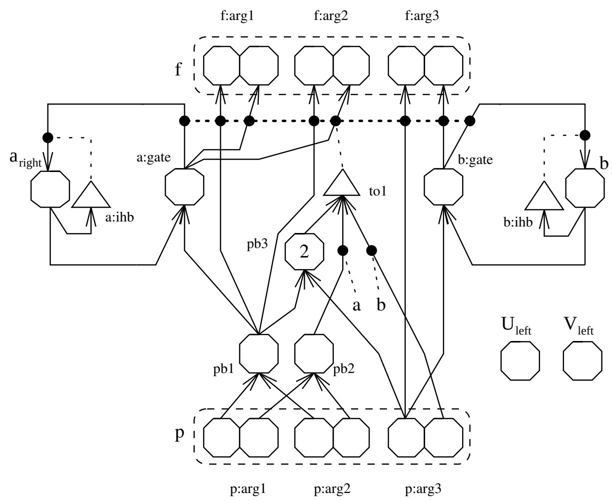

Figure 4: Encoding of the fact, $p(a, a, b)$

The fact *f:p(a,a,b)*, for instance, is encoded as shown in Figure 4. There shows the fact assembly f and its base predicate assembly p. Between them are the consistency checking sub-mechanisms which ensure constant and variable consistency between encoded fact and a query predicate during query processing. The nodes, *pb1* and *pb2*, are used to collect initial bindings from repeated constant arguments. The fact inhibitory links between *pb2* and *to1*, between $p{:}arg3_{right}$ and *to1*, are used for constant consistency checking. The nodes, *pb1*, $p{:}arg3_{left}$, *pb3*, and *to1* and related links between them 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 becoming active. Otherwise, the initial bindings will propagate from the base predicate assembly to the fact assembly. In this case, the rest of the intermediate mechanism will activate each argument node of the fact assembly and constant nodes according to the unification condition between the encoded fact and the posed query predicate. The nodes *a:gate* and *b:gate* are used to activate the right arguments nodes of the fact assembly and their corresponding constant node at the same time. The nodes *a:ihb* and *b:ihb* are used to make each constant node becomes active only once during a chain of inference. Once constant nodes a and b become active their activation makes associated *ihb* nodes active and, as a result, all inputs coming to these constant nodes will be inhibited.

As shown in the above example, the proposed fact encoding mechanism does not need different network configurations for yes-no and and wh-type query processing (as is the case in Shastri & Ajjanagadde's system). These two query procedures are merged into one fact encoding mechanism. This mechanism does not even require two separate stages of wh-type query processing because the result of unification is automatically represented on each argument node of the fact assembly during the query processing.

6 Performing Inference

Once a set of symbolic rules and facts are encoded into a network, the inference system can perform either forward chaining or backward chaining over the network.

Forward chaining is started by presenting a predicate to the system. The presented predicate has the form $pred\_name(t_1, t_2, \ldots, t_m)$, where t_i's are either constants or existentially quantified variables. Presenting a predicate to the system involves specifying the argument bindings between the presented predicate and its corresponding base predicate assembly in the network. Setting up the initial bindings is done as follows:

- assign each entity node (called a filler node) to each filler which appears in the input predicate
- activate each filler node and the corresponding argument of the base predicate in a unique phase in such a way that: if the filler is a constant, activate both the right π-btu element of the filler and the argument nodes; if the filler is a variable, activate both the left π-btu element of the filler and the argument nodes.

Once these initial variable bindings are set up using temporal synchrony, these bindings are then propagated to the consequent predicate assembly and, as a result, its arguments will represent the result of inference. In Figure 3, for instance, presenting the input fact, $p(a,U,U)$, to the network will set up initial variable bindings, $\{a/p{:}arg1,\ U/p{:}arg2,\ U/p{:}arg3\}$ by activating the filler nodes and their corresponding argument nodes in the following phases:

$$a_{right}[1],\ U_{left}[2],\ p{:}\{arg1[\phi,1],\ arg2[2,\phi],\ arg3[2,\phi]\}$$

where the numbers specify the particular phases. These activations propagate to the q predicate assembly and activate its argument nodes

$$q{:}\{arg1[2,1],\ arg2[2,1]\}$$

which represents the result of inference $\{q(a,a)\}$ and the result of unification during the inference $\{U/a\}$.

Backward chaining is also started by posing a query predicate to the system. The query predicate has the same form as that used in forward chaining. Posing a query predicate sets up the initial bindings between the posed query and the corresponding base predicate assembly (the consequent predicate assembly) in the same way as shown for forward chaining. The result of inference is then obtained by propagating these initial bindings to the antecedent predicate assembly and the associated fact assemblies if any. An associated fact assembly will only be activated if it matches with the query predicate posed. If the posed query contains variable fillers, the arguments of the activated fact assembly will represent the result of unification obtained from the inference. Therefore, after a query is posed to the system, the final result of backward chaining is obtained by observing the states of each fact assembly and its base predicate asembly in the network.

For example posing the query, $q(a,U,V)$, to the network shown in Figure 4 will set up initial variable bindings, $\{a/p{:}arg1,\ U/p{:}arg2,\ V/p{:}arg3\}$ by activating the filler nodes and their corresponding argument nodes in the following phases:

$$a_{right}[1],\ U_{left}[2],\ V_{left}[3],$$
$$p{:}\{arg1[\phi,1],\ arg2[2,\phi],\ arg3[3,\phi]\}$$

These initial bindings then propagate into the intermediate mechanism including consistency checking submechanism. Since the posed query does not violate consistency condition forced, these bindings propagate to the fact asembly through the rest of the intermediate mechanism. As a result the arguments of the fact assembly and their corresponding constant nodes will be activated as follows:

$$b_{right}[3],\ f{:}\{arg1[\phi,1],\ arg2[2,1],\ arg2[3,3]\}$$

This represents that the encoded fact, $f(a,a,b)$, matches with the posed query and the result of unification obtained from the second and third argument nodes of the fact predicate is $\{U/a, V/b\}$.

7 Concluding Remarks

Any connectionist knowledge-based system which aspires to imitate symbolic inference should be able to represent variables and constants and unification between them not only within a group of unifying arguments but also across many groups of unifying arguments. SHRUTI is a step in this direction but it does not provide a connectionist equivalent of a naming system for unbound variables, which is a fundamental requirement in all of the standard symbolic reasoning systems.

In the proposed inference system, both constant and variable fillers are represented using distinct phases. The use of a double π-btu elements for each argument node of a predicate and a fact assembly in encoding rules and facts allows the inference system easily to represent the result of inference and unification on each argument node. The various intermediate mechanisms adopted in the rule and fact encoding mechanism provide a way of representing not only a group of unifying arguments but also many groups of such arguments involving consistency checking. The modified temporal behaviour of a π-btu node provides a way of representing multiple variable bindings of an argument node within an oscillation cycle.

In conclusion, the proposed inference system provides a mechanism which translates a significant subset of Horn-clause into a connectionist representation which may be executed very efficiently (not only in a 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 structured terms as arguments of a predicate and recursion in rules. Currently, no connectionist system has provided a convincing solution to all these problems and to do so, in general, we would have to sacrifice one of the most attractive features of this class of system: its ability to guarantee an answer in finite time.

Acknowledgements

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

References

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

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

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

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