Combining a Connectionist Type Hierarchy with a Connectionist Rule-Based Reasoner*

D. R. Mani and **Lokendra Shastri**
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104, USA
shastri@cis.upenn.edu

Abstract

This paper describes an efficient connectionist knowledge representation and reasoning system that combines rule-based reasoning with reasoning about inheritance and classification within an *IS-A* hierarchy. In addition to a type hierarchy, the proposed system can encode generic facts such as 'Cats prey on birds' and rules such as 'if x preys on y then y is scared of x' and use them to infer that Tweety (who is a Canary) is scared of Sylvester (who is a cat). The system can also encode qualified rules such as 'if an *animate* agent walks into a *solid object* then the agent gets hurt'. The proposed system can answer queries in time that is only *proportional* to the *length* of the shortest derivation of the query and is *independent* of the *size* of the knowledge base. The system maintains and propagates variable bindings using temporally synchronous – i.e., in-phase – firing of appropriate nodes.

Introduction

In [Shastri & Ajjanagadde 1990a, Shastri & Ajjanagadde 1990b, Ajjanagadde & Shastri 1991], Ajjanagadde and Shastri have described a solution to the variable binding problem [Lange & Dyer 1989, Smolensky 1987] and shown that the solution leads to the design of a connectionist reasoning system that can represent systematic knowledge involving n-ary predicates and *variables*, and perform a broad class of reasoning with extreme efficiency. The time taken by the reasoning system to draw an inference is only proportional to the *length* of the chain of inference and is independent of the number of rules and facts encoded by the system. The reasoning system maintains and propagates variable bindings using temporally synchronous – i.e., in-phase – firing of appropriate nodes. The solution to the variable binding problem allows the system to maintain and propagate a large number of bindings *simultaneously* as long as the number of *distinct* entities participating in the bindings during any given episode of reasoning, remains bounded. Reasoning in the proposed system is the transient but systematic flow of *rhythmic* patterns of activation, where each *phase* in the rhythmic pattern corresponds to a distinct *constant* involved in the reasoning process and where variable bindings are represented as the synchronous firing of appropriate argument and constant nodes. A fact behaves as a temporal pattern matcher that becomes 'active' when it detects that the bindings corresponding to it are present in the system's pattern of activity. Finally, rules are interconnection patterns that propagate and transform rhythmic patterns of activity.[1]

In this paper we describe how the above reasoning system may be combined with an *IS-A* hierarchy. Such an integration allows the occurrence of types (categories) as well as instances in rules, facts, and queries. This has the following interesting consequences. First, the reasoning system can combine rule-based reasoning with inheritance and classification. For example, such a system can infer that 'Tweety is scared of Sylvester', based on the generic fact 'Cats prey on birds', the rule 'If x preys on y then y is scared of x and the *IS-A* relations 'Sylvester is a Cat' and 'Tweety is a Bird'.[2] Second, the integrated system can use category information to *qualify* rules by specifying restrictions on the type of argument fillers. An example of such a rule is: $\forall x$*:animate, y:solid-obj* [*walk-into(x,y)* $\Rightarrow$ *hurt(x)*], which specifies that the rule is applicable only if the two arguments of 'walk-into' are of the type 'animate' and 'solid-object', respectively.

An overview of the rule-based reasoning system is followed by a description of the *IS-A* hierarchy realization and its interface with the reasoning system. A detailed discussion of the reasoning system and the type hierarchy interface may be found in [Shastri & Ajjanagadde 1990b] and [Mani & Shastri 1991] respectively.

*This work was supported by NSF grant IRI 88-05465 and ARO grant ARO-DAA29-84-9-0027.

[1] It may be worth stating that the system does *not* require a central controller or a global clock.

[2] Observe that this kind of reasoning combines 'relational inheritance' [Fahlman 1979] with rule-based reasoning: While relational inheritance can support the inference 'Sylvester preys on Tweety' by using the *IS-A* relationships on the generic fact 'Cats prey on birds', it cannot support the inference 'Tweety is scared of Sylvester', because doing so also requires the use of the rule 'If x preys on y then y is scared of x'.

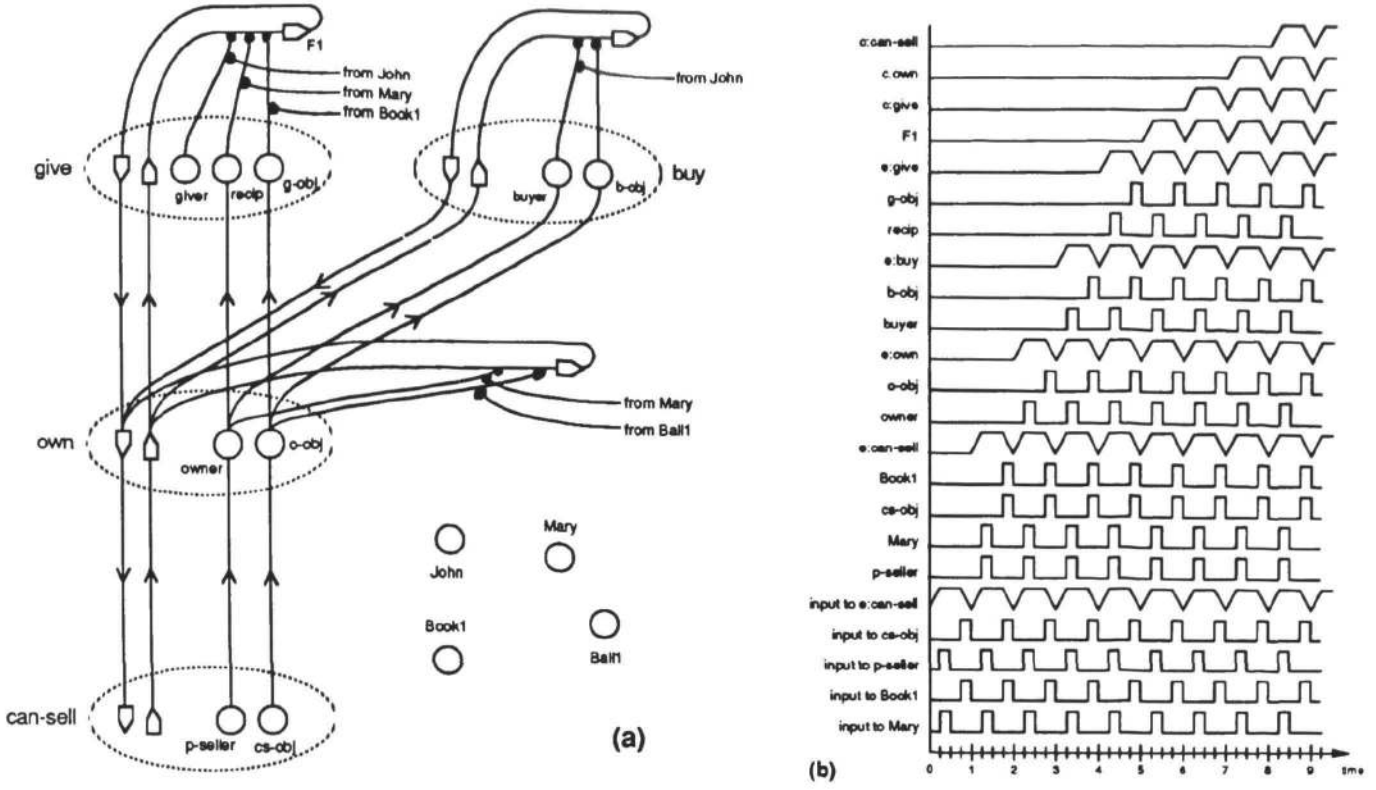

Figure 1: (a) Encoding rules and facts. (b) Activation trace for the query *can-sell(Mary,Book1)?*

The rule-based reasoning system

Fig. 1a illustrates how long-term knowledge is encoded in the rule-based reasoning system. The network encodes the following *rules* and *facts*: i) $\forall x, y, z$ [*give(x,y,z)* $\Rightarrow$ *own(y,z)*], ii) $\forall x, y$ [*buy(x,y)* $\Rightarrow$ *own(x,y)*], iii) $\forall x, y$ [*own(x,y)* $\Rightarrow$ *can-sell(x,y)*], iv) *give(John,Mary,Book1)*, v) *buy(John,x)*, and vi) *own(Mary,Ball1)*.

The encoding makes use of two types of nodes: ρ-btu nodes (depicted as circles) and τ-and nodes (depicted as pentagons). The computational behavior of these nodes is as follows: A ρ-btu is a phase-sensitive binary threshold unit. When such a node becomes active, it produces an oscillatory output in the form of a pulse train that has a period π and pulse width ω. The timing (or the *phase*) of the pulse train produced by a ρ-btu node depends on the phase of the input to the node. A τ-and node acts like a *temporal* AND node. Such a node also oscillates with the same frequency as a ρ-btu node except that it becomes active only if it receives *uninterrupted* activation over a whole period of oscillation. Furthermore, the width of the pulses produced by a τ-and node equals π.[3] The maximum number of distinct entities that may participate in the reasoning process equals π/ω (assume integer divide). The encoding also makes use of *inhibitory modifiers* – links that impinge upon and inhibit other links. A pulse propagating along an inhibitory modifier will block a pulse propagating along the link it impinges upon. In Fig. 1a, inhibitory modifiers are shown as links ending in dark blobs.

Each constant in the domain is encoded by a ρ-btu node. An n-ary predicate is encoded by a pair of τ-and nodes and n ρ-btu nodes, one for each of the n arguments. One of the τ-and nodes is referred to as the *enabler* and the other as the *collector*. As a matter of convention, an *enabler* always points upwards and is named *e:<predicate-name>*. A *collector* always points downwards and is named *c:<predicate-name>*. The *enabler e:P* of a predicate P becomes active whenever the system is being queried about P. Such a query may be posed by an external process or by the system itself during an episode of reasoning. On the other hand, the system activates the *collector c:P* of a predicate P whenever the system wants to assert that the current dynamic bindings of the arguments of P are consistent with the knowledge encoded in the system. A rule is encoded by connecting the *collector* of the antecedent predicate to the *collector* of the consequent predicate, the *enabler* of the consequent predicate to the *enabler* of the antecedent predicate, and by connecting the arguments of the consequent predicate to the arguments of the antecedent predicate in accordance with the correspondence between these arguments specified in the rule. A fact is encoded using a τ-and node that receives an input from the enabler of the associated predicate. This input is modified by inhibitory modifiers from the argument nodes of the associated predicate. If an argument is bound to a constant in the fact then the modi-

[3] Later we will introduce a third type of node, namely the τ-or node. A τ-or node becomes active on receiving *any* activation but its output is like that of a τ-and node.

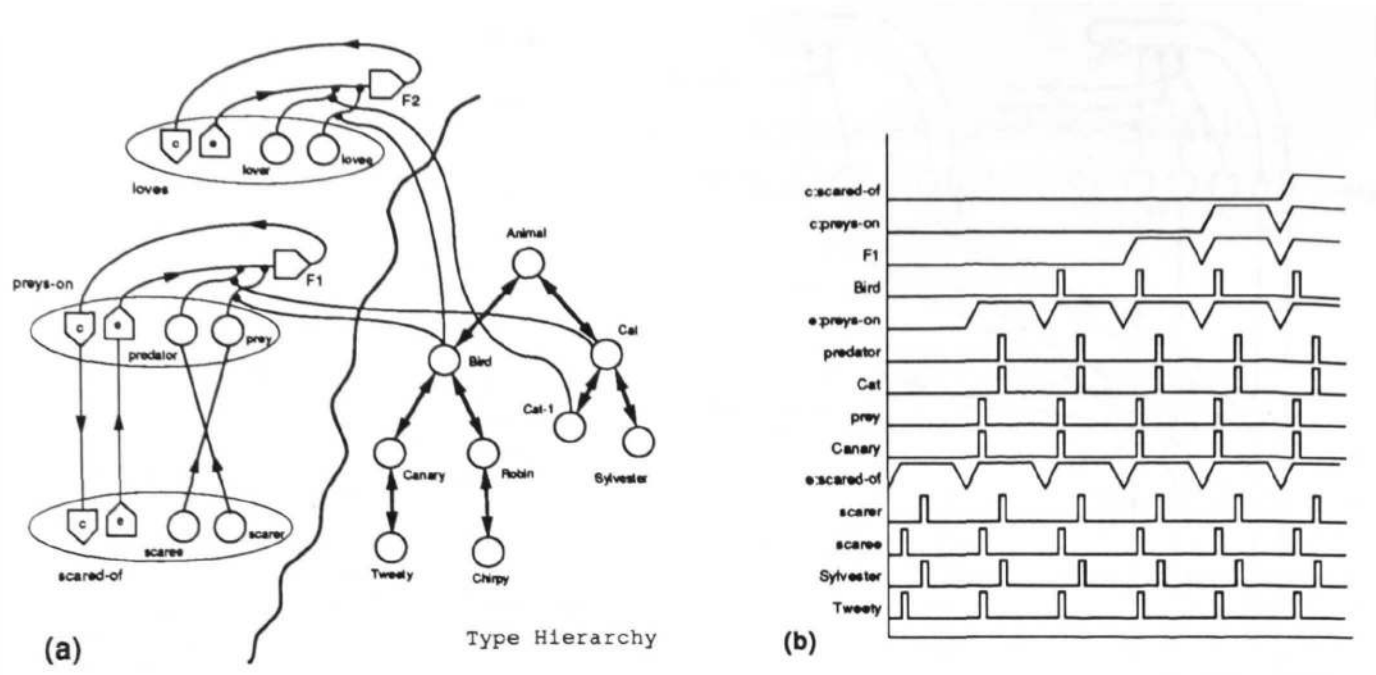

Figure 2: (a) An example network. (b) Trace of spreading activation for the query *scared-of(Tweety,Sylvester)?*

fier from such an argument node is in turn modified by an inhibitory modifier from the appropriate constant node. The output of the τ-and node is connected to the *collector* of the associated predicate (refer to the encoding of the fact *give(John,Mary,Book1)* and *buy(John,x)* in Fig. 1a.)

Inference Process

Posing a query to the system involves specifying the query predicate and the argument bindings specified in the query. In the proposed system this is done by simply activating the relevant nodes in the manner described below. Choose an arbitrary reference point in time – say, t_0 – for initiating the query. We assume that the system is in a quiescent state just prior to t_0. The query predicate is specified by activating the *enabler* of the query predicate, with a pulse train of width and periodicity π starting at time t_0.

The argument bindings specified in the query are communicated to the network as follows: Suppose the argument bindings in the query involve n distinct constants $c_1, ..., c_n$. With each of these n constants, associate a delay δ_i such that no two delays are within ω of one another and the longest delay is less than $\pi - \omega$. Each of these delays may be viewed as a distinct *phase* within the period t_0 and $t_0 + \pi$. Now the argument bindings of a constant c_i are indicated to the system by providing an oscillatory pulse train of pulse width ω and periodicity π starting at $t_0 + \delta_i$, to c_i and all arguments to which c_i is bound. This is done for each constant c_i $(1 \leq i \leq n)$ and amounts to representing argument bindings by the *in-phase or synchronous activation of the appropriate constant and argument nodes.*

We illustrate the reasoning process with the help of an example. Consider the query *can-sell(Mary,Book1)?* (i.e., Can Mary sell Book1?) The query is posed by i) activating the *enabler* node *e:can-sell* ii) activating *Mary* and *p-seller* in the same phase (say, phase-1), and iii) activating *Book1* and *cs-obj* in some other phase (say, phase-2). As a result of these inputs, *Mary* and *p-seller* will fire synchronously in phase-1 of every period of oscillation, while *Book1* and *cs-obj* will fire synchronously in phase-2 of every period of oscillation. The node *e:can-sell* will also oscillate and generate a pulse train of periodicity and pulse width π (Fig. 1b). The activations from the arguments *p-seller* and *cs-obj* reach the arguments *owner* and *o-obj* of the predicate *own*, and consequently, starting with the second period of oscillation, *owner* and *o-obj* become active in phase-1 and phase-2, respectively. At the same time, the activation from *e:can-sell* activates *e:own*. The newly created dynamic bindings for the arguments of *own*, in conjunction with the activation of *e:own* can be thought of as encoding the query *own(Mary,Book1)?* The τ-and node associated with the fact *own(Mary, Ball1)* does not match the query and remains inactive. As the activation propagates from the arguments of *own* to the arguments of *buy* and *give* (Fig. 1b), new bindings for the arguments of *give* and *buy* get established which, in effect, encode two new queries: *give(x,Mary,Book1)?* (i.e., Did *someone* give Mary Book1?), and *buy(Mary,Book1)?* The τ-and node F1, associated with the fact *give(John,Mary,Book1)* becomes active as a result of the uninterrupted activation it receives from *e:give*. Observe that the inhibitory modifiers from *recip* and *g-obj* incident on the link from *e:own* to F1 are blocked by the in-phase inputs from *Mary* and *Book1*, respectively. The activation from F1 causes *c:give*, the *collector* of *give*, to become active and the output from *c:give* in turn causes *c:own* to become active and transmit an output to *c:can-sell*. Consequently *c:can-sell*, the *collector* of the query predicate *can-sell*, becomes active indicating an affirmative answer to the query *can-sell(Mary,Book1)?* (Fig. 1b).

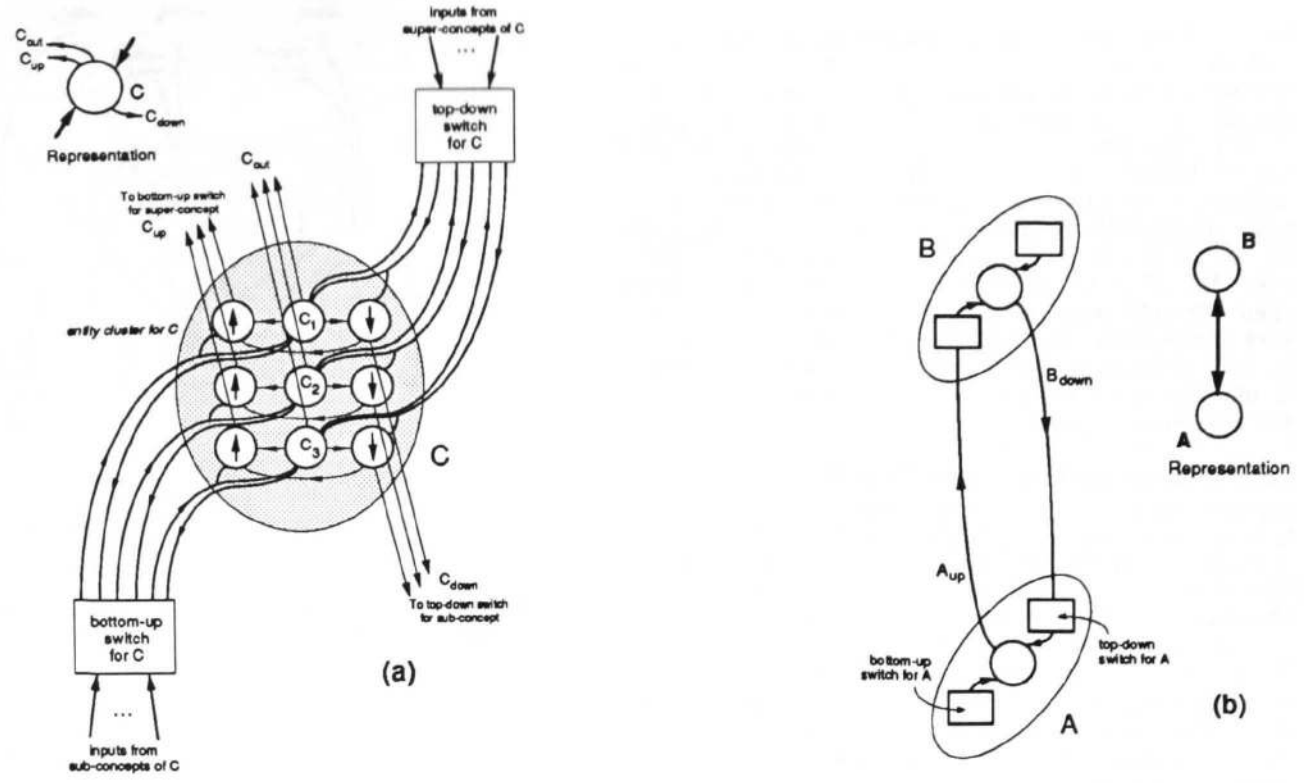

Figure 3: (a) Structure of the entity cluster for C, and its interaction with the bottom-up and top-down switches. The $\uparrow$ and $\downarrow$ nodes have a threshold $\theta = 2$. The multiple instantiation constant, $k = 3$. (b) Encoding of the *is-a* relation *is-a(A,B)*. A bundle of k wires is represented by a single link.

Combining the rule-based reasoner with an *IS-A* hierarchy

Fig. 2a gives an overview of the combined reasoning system. The rule-based part of the network encodes the rule $\forall x, y$ [*preys-on(x,y)* $\Rightarrow$ *scared-of(y,x)*] (i.e., if x preys on y, then y is scared of x), and the facts $\forall x$*:Cat,* y*:Bird preys-on(x,y)* and $\exists x$*:Cat* $\forall y$*:Bird loves(y,x)*. The former is equivalent to *preys-on(Cat,Bird)* and amounts to 'Cats prey on Birds'. The latter amounts to 'there is a cat that loves all birds'. The network on the right encodes the *IS-A* relationships: *is-a(Bird, Animal)*, *is-a(Cat, Animal)*, *is-a(Robin, Bird)*, *is-a(Canary, Bird)*, *is-a(Tweety, Canary)*, *is-a(Chirpy, Robin)*, and *is-a(Sylvester, Cat)*.

Facts involving typed variables are encoded in the following manner: A typed, universally quantified variable is treated as being equivalent to its type. Thus $\forall x$*:Cat,* y*:Bird preys-on(x,y)* is encoded as *preys-on(Cat,Bird)*. A typed, existentially quantified variable is encoded using a unique subconcept of the associated type. Thus in Fig. 2a, $\exists x$*:Cat* $\forall y$*:Bird loves(x,y)* is encoded as *loves(Cat-1,Bird)*, where *Cat-1* is a unique instance of *Cat*. For now let us assume that i) each concept (type or instance) is encoded as a ρ-btu node ii) each conceptual *IS-A* relationship such as *is-a(A,B)* is encoded using two connectionist links – a bottom-up link from A to B and a top-down link from B to A, and iii) the top-down and bottom-up links can be enabled selectively by built-in control mechanisms (missing details are provided below).

The time course of activation for the query *scared-of(Tweety,Sylvester)?* is given in Fig. 2b. The query is posed by turning on *e:scared-of* and activating the nodes *Tweety* and *Sylvester* in synchrony with the first and second arguments of *scared-of*, respectively. The bottom-up links emanating from *Tweety* and *Sylvester* are also enabled. The net result of spreading activation (Fig. 2b), in the conceptual hierarchy and the rule-base is that the query *scared-of(Tweety,Sylvester)?* is transformed into the query *preys-on(Cat,Bird)?* The latter query matches the stored fact *preys-on(Cat,Bird)* and leads to the activation of *c:preys-on*. In turn, *c:scared-of* becomes active and signals an affirmative answer to the query.

Two technical problems

There are two technical problems that must be solved in order to integrate the conceptual hierarchy and the rule-based component. First, the encoding of the *IS-A* hierarchy should be capable of representing multiple instantiations of a concept. For example, in the query discussed above, we would like the network's state of activation to represent both 'the animal Tweety' and 'the animal Sylvester'. This is problematic because the node *Animal* cannot be in synchrony with both *Tweety* and *Sylvester* at the same time. Second, the encoding must provide *built-in* mechanisms for controlling the direction of propagating activation in the *IS-A* hierarchy so

as to correctly deal with queries containing existentially and universally quantified variables. Thus i) Activation originating from an instance or a concept that corresponds to a universally quantified variable in the query should propagate upwards to all its ancestors, and ii) If the *IS-A* hierarchy is a taxonomy, then activation originating from a concept C that corresponds to an existentially quantified variable in the query should propagate to the ancestors as well as descendents of C. If however, the *IS-A* hierarchy permits multiple inheritance then the activation must *also* propagate to the ancestors of the descendents of C. The following solution to these problems does not require an external controller to monitor and control the nodes in the network, during the reasoning process.

Implementing the Type Hierarchy

Each entity (i.e., type or instance) C, is represented by a group of nodes called the *entity cluster* for C. Such a cluster is organized as shown in Fig. 3a. The entity cluster for C has k *banks* of ρ-btu nodes, where k, the *multiple instantiation constant*, refers to the number of dynamic instantiations a concept can accommodate. Each *bank* C_{B_i} consists of three ρ-btu nodes: C_i, $C_{i\uparrow}$, $C_{i\downarrow}$. Each C_i represents a distinct (dynamic) instantiation of C. If this instantiation is in phase ρ, then, C_i fires in phase ρ. The *relay nodes* $C_{i\uparrow}$ and $C_{i\downarrow}$ control the *direction of propagation* of the activation represented by C_i. The $C_{i\uparrow}$ and $C_{i\downarrow}$ nodes have a threshold $\theta = 2$. As shown in Fig. 3a, C_i is connected to both $C_{i\uparrow}$ and $C_{i\downarrow}$. $C_{i\downarrow}$ is linked to $C_{i\uparrow}$, but not vice versa. Directional control of propagating activation is exercised using a suitable modification of the *relay-node* scheme discussed in [Shastri 1988].

Every entity C is associated with two *switches* – a *top-down* switch and a *bottom-up* switch. The switches, both of which are identical in structure, control the flow of activation in the type hierarchy. Each switch has k outputs. $Output_i$ from the bottom-up switch connects to C_i and $C_{i\uparrow}$ while the corresponding output from the top-down switch goes to the C_i and $C_{i\downarrow}$ nodes, $1 \leq i \leq k$. There is also a feedback from the C_i nodes to both the switches (See Fig. 3a and Fig. 4.)

The interaction between the switches and the entity cluster (Fig. 3a) brings about efficient and *automatic dynamic allocation of banks* in an entity cluster, by ensuring that:

- Activation is channeled to the entity cluster banks only if the entity cluster can accommodate more instantiations; the maximum number of instantiations is therefore limited to k.
- Each C_i picks up a *unique* phase; thus new instantiations are always in a phase not already represented in the entity cluster.

The architecture of the switch (with $k = 3$) is illustrated in Fig. 4. The k ρ-btu nodes, $S_1, \ldots, S_k$, with their associated τ-or nodes form the basic components of the switch. Every input to the switch makes two connections – one excitatory and one inhibitory – to each of $S_2, \ldots, S_k$; as a result of these excitatory-inhibitory connections, all these nodes are disabled to begin with, and cannot respond to incoming activation. Input activation will have an effect only on the S_1 node, since the

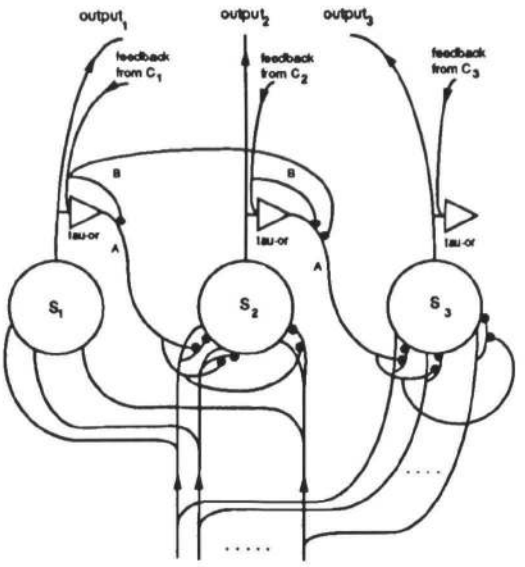

Figure 4: Architecture of the switch. The multiple instantiation constant $k = 3$.

inputs to the switch directly connect to S_1 (Fig. 4.) In keeping with the behavior of ρ-btu nodes, S_1 becomes active in response to the first available input and continues to fire in phase with that input as long it remains active. As S_1 goes active, the τ-or node associated with S_1 turns on, thereby enabling S_2. Inhibitory feedback from C_1 ensures that S_2 is *not* enabled during the phase ρ in which C_1 is firing[4]. Thus S_2 selects and starts firing in a phase *other than* ρ. Once S_2 has made its selection, S_3 gets its turn, and so on.

As instantiations are deputed to the entity cluster, the ρ-btu nodes in the switch are progressively enabled from left to right. If $C_1, \ldots, C_{i-1}$ are firing in phases $\rho_1, \ldots, \rho_{i-1}$, then S_i always picks a distinct phase $\rho \notin \{\rho_1, \ldots, \rho_{i-1}\}$, since inputs in phases $\rho_1, \ldots, \rho_{i-1}$ are inhibited by the feedback links from $C_1, \ldots, C_{i-1}$. At any stage, if C_i, $1 \leq i \leq k$, picks up activation channeled by the *other* switch, feedback from C_i into the τ-or node associated with S_i causes S_{i+1} to be enabled, even though S_i has not picked a phase. This ensures that at most k instantiations are selected *jointly* by the bottom-up and top-down switches; hence, only k instantiations can be channeled to C, at worst.

A fact of the form *is-a(A,B)* is represented by (Fig. 3b): (i) connecting the $A_{i\uparrow}, i = 1, \ldots, k$ nodes to the bottom-up switch for B; (ii) connecting the $B_{i\downarrow}, i = 1, \ldots, k$ nodes to the top-down switch for A.

Consider a concept C in the type hierarchy. Suppose C_i receives activation from the bottom-up switch in phase ρ. C_i starts firing in synchrony with this activation. The $C_{i\uparrow}$ node (with threshold $\theta = 2$) is now receiving *two* inputs in this phase (from the bottom-up switch and from C_i; see Fig. 3a) and begins to fire in phase ρ. This causes activation in phase ρ to eventually

[4] In general, C_i could receive input in *two* phases – one from the bottom-up switch for C, and another from its top-down switch. C_i being a ρ-btu node, picks one of these phases to fire in.

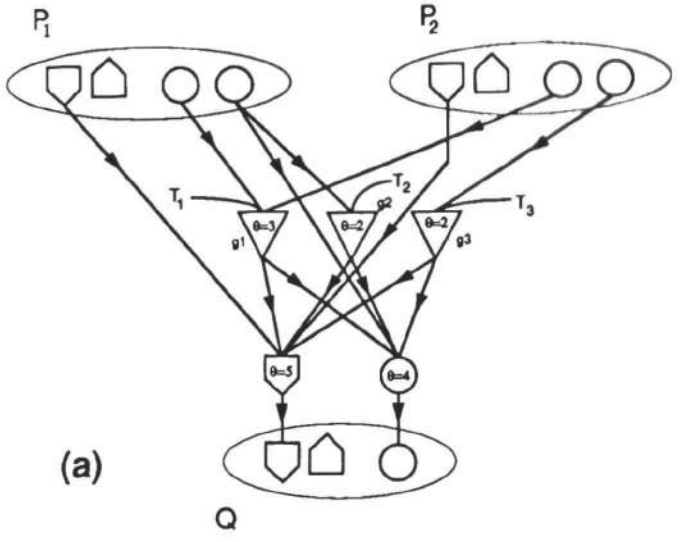

Figure 5: Network encoding the rule $\forall x{:}T_1,\ y{:}T_2\ \exists z{:}T_3$ [$P_1(x,y)$ & $P_2(x,z) \Rightarrow Q(y)$] in a forward reasoning system. Links from T_1, T_2 and T_3 are actually bundles of k wires carrying the k instantiations of these constants.

spread to the super-concept of C. Hence, any upward traveling activation continues to travel upward – which is the required behavior when C is associated with a universal typed variable. Similarly, when C_i receives activation from the top-down switch in phase ρ, both C_i and $C_{i\downarrow}$ become active in phase ρ. $C_{i\uparrow}$ follows suit, because of the link from $C_{i\downarrow}$ to $C_{i\uparrow}$, so that the whole bank C_{B_i} now fires in phase ρ. This mechanism allows a concept associated with an existential typed variable to eventually spread its activation to its ancestors, descendents and ancestors of descendents.

Typed Variables in Rules

The type hierarchy can be used to impose type restrictions on variables occurring in rules, for both forward and backward reasoning systems. To utilize this feature, we need to modify the implementation of rules: In a *forward reasoning system*, the rule is encoded by introducing a τ-or node to perform the type checking for the argument under question. For example, in Fig. 5, which encodes the rule $\forall x{:}T_1,\ y{:}T_2\ \exists z{:}T_3$ [$P_1(x,y)$ & $P_2(x,z) \Rightarrow Q(y)$], g_3 would turn on if and only if the second argument of P_2 and T_3 are in synchrony – which is to say that the argument is bound to an object of type T_3. In the forward reasoner, typed variables are allowed only in the antecedent of the rule.

In a *backward reasoner*, the strategy is similar, except that i) type checking for a typed universally quantified variable is enforced by an inhibitory link from the concept representing the type of the concerned argument; ii) For a typed, existentially quantified variable, an inhibitory link derived from a unique subconcept of the associated type performs the type enforcement – in a manner similar to the interpretation of a typed existential variable in a fact [Mani & Shastri 1991]. In the backward reasoner, typed variables are allowed only in the consequent. Both in the forward and backward reasoners, a rule fires only if *all typed arguments are firing in synchrony with their respective types.*

Conclusions

Adding a type hierarchy allows the connectionist reasoning system to handle rules, facts, and queries with typed variables. The simulator described in [Mani 1990] has been used to test the system. Several extensions to the system proposed here are being investigated. The current system assumes that any fact or query with both existentially and universally quantified variables is such that all the universal quantifiers are within the scope of the existential quantifiers. Work is being done on handling more general forms of facts and queries. We are also working on the design of an expanded system that would allow property-value attachments to concepts [Shastri 1988]. We also wish to combine the *IS-A* hierarchy with a reasoning system that allows multiple instantiation of predicates. Lastly, we hope to combine a forward and backward reasoner that can make use of both long-term and dynamic (temporary) facts during reasoning.

References

[Ajjanagadde & Shastri 1991] Ajjanagadde, V., and Shastri, L. 1991. Rules and variables in neural nets. To appear in *Neural Computation* 3(1).

[Fahlman 1979] Fahlman, S. 1979. *NETL: A System for Representing Real-World Knowledge.* Cambridge, Mass.: MIT Press.

[Lange & Dyer 1989] Lange, T. E., and Dyer, M. G. 1989. High-level inferencing in a connectionist network. *Connection Science*, 1(2):181-217.

[Mani 1990] Mani, D. R. 1990. *Using the Connectionist Rule-Based Reasoning System Simulator.*

[Mani & Shastri 1991] Mani, D. R., and Shastri, L. Combining a Type Hierarchy with a Connectionist Rule-Based Reasoner. Technical Report MS-CIS-91-33, Department of Computer and Information Science, Univ. of Pennsylvania, May 1991.

[Shastri 1988] Shastri, L. 1988. *Semantic networks: An evidential formulation and its connectionist realization.* Los Altos: Morgan Kaufman.

[Shastri & Ajjanagadde 1990a] Shastri, L., and Ajjanagadde, V. 1990. An optimally efficient limited inference system. In *Proceedings of AAAI-90, the Twelfth National Conference of the American Association of Artificial Intelligence*, 563-570. Cambridge Mass.: American Association for Artificial Intelligence.

[Shastri & Ajjanagadde 1990b] Shastri, L., and Ajjanagadde, V. 1990. From Simple Associations to Systematic Reasoning: A Connectionist Representation of Rules, Variables and Dynamic Bindings. Technical Report MS-CIS-90-05, Department of Computer and Information Science, Univ. of Pennsylvania.

[Smolensky 1987] Smolensky, P. 1987. On variable binding and the representation of symbolic structures in connectionist systems. Technical Report CU-CS-355-87, Department of Computer Science, Univ. of Colorado at Boulder.