May 1991

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

Lokendra Shastri
University of Pennsylvania

D. R. Mani
University of Pennsylvania

Follow this and additional works at: http://repository.upenn.edu/cis_reports

Recommended Citation
http://repository.upenn.edu/cis_reports/861


This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_reports/861
For more information, please contact repository@pobox.upenn.edu.
Combining a Connectionist Type Hierarchy with a Connectionist Rule-Based Reasoner

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 Sylvestor (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.

Comments
Combining A Connectionist Type Hierarchy
With A Connectionist Rule-Based Reasoner

MS-CIS-91-33
LINC LAB 201

Lokendra Shastri
D.R. Mani

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

May 1991
Combining a Connectionist Type Hierarchy with a Connectionist Rule-Based Reasoner*

D. R. Mani        Lokendra Shastri

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104, USA
(215) 898-2661
shastri@central.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 z preys on y then y is scared of z' and use them to infer that Tweety (who is a Canary) is scared of Sylvestor (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.

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

In [8, 7, 1], Ajjanagadde and Shastri have described a solution to the variable binding problem ([4], [9]) 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

This report describes 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:

- 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

As a byproduct of this capability, the system can encode facts with typed variables (for example, ∀x:Cat, y:Bird preys-on(x,y) and ∃x:Cat, ∀y:Bird loves(x,y) which mean ‘All cats prey on all birds’ and ‘there is a cat which loves all birds’ respectively) and can answer queries with typed variables (like ∃x:Cat ∀y:Bird scared-of(y,x)?, i.e. ‘is there a cat such that all birds are scared of it?’).

- The integrated system can use category information to qualify rules by specifying restrictions on the type of argument fillers. Examples of such rules are:

  \[ \forall x:animate, y:solid\text{-}object \text{ walk\text{-}into}(x,y) \Rightarrow \text{hurt}(x) \]
  \[ \forall x:animate \exists y:animate \text{ predator}(z) \Rightarrow \text{preys\text{-}on}(z,y) \]

The first rule is applicable only if the two arguments of ‘walk\text{-}into’ are of the type ‘animate’ and ‘solid\text{-}object’, respectively; the second rule is applicable only if both the variables are of type ‘animate’.

Section 2 provides a brief overview of the rule-based reasoning system while Section 3 provides an overview of the new system. The next three sections describe the realization of the IS-A hierarchy and its interface with the reasoning system. The last section concludes the paper and indicates several extensions. A detailed discussion of the reasoning system may be found in [8].

2 The rule-based reasoning system

Fig. 1a illustrates how long-term knowledge is encoded in the rule-based reasoning system. The network shown in Fig. 1a encodes the following facts and rules:

---

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

²Observe that this kind of reasoning combines Fahhman’s ‘relational inheritance’ ([2]) 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’.

---

2

2
Figure 1: (a) An example encoding of rules and facts. (b) Activation trace for the query can-sell(Mary, Book1)?.

\[
\forall x, y, z \ givex(y, z) \Rightarrow own(x, y) \\
\forall x, y \ buy(x, y) \Rightarrow own(x, y) \\
\forall x, y \ own(x, y) \Rightarrow can-sell(x, y) \\
givex(John, Mary, Book1) \\
buy(John, x) \\
\text{own}(Mary, Ball1).
\]

The encoding makes use of two types of nodes. These are \(p\)-btu nodes (depicted as circles) and \(\tau\)-and nodes (depicted as pentagons). The computational behavior of these nodes is as follows: A \(p\)-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 \(\pi\) and pulse width \(\omega\). The timing (or the phase) of the pulse train produced by a \(p\)-btu node depends on the phase of the input to the node. A \(\tau\)-and node acts like a \textit{temporal} and node. Such a node also oscillates with the same frequency as a \(p\)-btu node except that it becomes active only if it receives \textit{uninterrupted} activation over a whole period of oscillation. Furthermore, the width of the pulses produced by a \(\tau\)-and node equals \(\pi/\omega\). The maximum number of distinct entities that may participate in the reasoning process equals \(\pi/\omega\) (assume integer divide). The encoding also makes use of \textit{inhibitory modifiers}. An inhibitory modifier is a link that impinges upon and inhibits another link. Thus a pulse propagating along an inhibitory modifier will block the propagation of a pulse propagating along the link it impinges upon. In Fig. 1a, inhibitory modifiers are shown as links ending in dark blobs.

\[\text{Later we will introduce a third type of node, namely the } \tau\text{-or node. A } \tau\text{-or node becomes active on receiving any activation but its output is like that of a } \tau\text{-and node.}\]
Each constant in the domain is encoded by a $\rho$-btu node. An n-ary predicate is encoded by a pair of $\tau$-and nodes and $n \rho$-btu nodes, one for each of the $n$ arguments. One of the $\tau$-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:\text{<predicate-name>}. A collector always points downwards and is named $c:\text{<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 $\tau$-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 modifier from such an argument node is in turn modified by an inhibitory modifier from the appropriate constant node. The output of the $\tau$-and node is connected to the collector of the associated predicate (refer to the encoding of the fact $\text{give}(\text{John}, \text{Mary}, \text{Book}1)$ and $\text{buy}(\text{John}, x)$ in Fig. 1a.)

### 2.1 The 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. Let us choose an arbitrary point in time – say, $t_0$ – as our point of reference 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 $\pi$ starting at time $t_0$.

The argument bindings specified in the query are communicated to the network as follows: Let the argument bindings in the query involve $k$ distinct constants: $c_1, ..., c_k$. With each of these $k$ constants, associate a delay $\delta_i$ such that no two delays are within $\omega$ 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 $\omega$ and periodicity $\pi$ starting at $t_0 + \delta_i$, to $c_i$ and all arguments to which $c_i$ is bound. This is done for each constant $c_i$ ($1 \leq i \leq k$) 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 $\text{can-sell}(\text{Mary}, \text{Book}1)$? (i.e., Can Mary sell Book1?) This query is posed by providing inputs to the constants $\text{Mary}$ and $\text{Book}1$, the arguments $\text{p-seller}$, $\text{cs-obj}$ and the enabler $e:\text{can-sell}$ as shown in Fig. 1b. $\text{Mary}$ and $\text{p-seller}$ receive in-phase activation and so do $\text{Book}1$ and $\text{cs-obj}$. Let us refer to the phase of activation of $\text{Mary}$ and $\text{Book}1$ as phase-1 and phase-2 respectively. As a result of these inputs, $\text{Mary}$ and $\text{p-seller}$ will fire synchronously in phase-1 of every period of oscillation, while $\text{Book}1$ and $\text{cs-obj}$ will fire synchronously in phase-2 of every period of oscillation. The node $e:\text{can-sell}$ will also oscillate and generate a pulse train of periodicity and pulse width $\pi$. The activations from the arguments $\text{p-seller}$ and $\text{cs-obj}$ reach the arguments $\text{owner}$ and $\text{o-obj}$ of the predicate $\text{own}$, and consequently, starting with the second period of oscillation, $\text{owner}$ and $\text{o-obj}$ become active in phase-1 and phase-2, respectively. At the same time, the activation from $e:\text{can-sell}$ activates $e:\text{own}$. The system has essentially, created dynamic bindings for the arguments of predicate $\text{own}$. $\text{Mary}$ has been bound to the argument $\text{owner}$, and $\text{Book}1$ has been bound to the argument $\text{own-object}$. These newly created bindings in conjunction with the activation of $e:\text{own}$ can be thought of as encoding the query $\text{own}(\text{Mary}, \text{Book}1)$? (i.e., 'Does Mary own Book1?')! The $\tau$-and node associated with the fact $\text{own}(\text{Mary}, \text{Ball}1)$ does not match the query and remains inactive. The activations from $\text{owner}$ and $\text{o-obj}$ reach the arguments $\text{recip}$ and $\text{g-obj}$ of $\text{give}$, and $\text{buyer}$ and $\text{b-obj}$ of $\text{buy}$ respectively. Thus beginning with the third period of oscillation, arguments $\text{recip}$

---

4We assume backward reasoning.
and \textit{buyer} become active in phase-1, while arguments \textit{g-obj} and \textit{b-obj} become active in phase-2. In essence, the system has created new bindings for the predicates \textit{can-sell} and \textit{buy} that can be thought of as encoding two new queries: \textit{give}(x, Mary, Book1)? (i.e., 'Did someone give Mary Book1?'), and \textit{buy}(Mary, Book1)?. Observe that now the \(r\)-and node associated with the fact \textit{give}(John, Mary, Book1) (this is the \(r\)-and node labeled F1 in Fig. 1a), becomes active as a result of the uninterrupted activation from \textit{e:give}. The inhibitory inputs from \textit{recip} and \textit{g-obj} are blocked by the in-phase inputs from \textit{Mary} and \textit{Book1}, respectively. The activation from this \(r\)-and node causes \textit{c:give}, the collector of \textit{give}, to become active and the output from \textit{c:give} in turn causes \textit{c:own} to become active and transmit an output to \textit{c:can-sell}. Consequently, \textit{c:can-sell}, the collector of the query predicate \textit{can-sell}, becomes active resulting in an affirmative answer to the query \textit{can-sell}(Mary, Book1)? (refer to Fig. 1b).

### 3 Combining the rule-based reasoner with an hierarchy: An Overview

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

\[
\forall x:Cat, y:Bird \ preys-on(x,y) \\
\exists x:Cat \forall y:Bird \ loves(x,y).
\]

The former fact is equivalent to \textit{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 \textit{IS-A} relationships:

\[
\begin{align*}
is-a(Bird, Animal) \\
is-a(Cat, Animal) \\
is-a(Robin, Bird) \\
is-a(Canary, Bird) \\
is-a(Chirpy, Robin) \\
is-a(Tweety, Canary) \\
is-a(Sylvester, Cat).
\end{align*}
\]

#### Interpreting Facts

Facts involving typed variables are encoded in the following manner:

- A typed, universally quantified variable is treated as being equivalent to its type. Also, any entity directly specified in a fact is treated as a substitute for a typed universal variable.\(^5\) Thus \(\forall x:Cat, y:Bird \ preys-on(x,y), \forall x:Cat \ preys-on(x,Bird)\) and \(\forall y:Bird \ preys-on(Cat,y)\) are all encoded as \textit{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 \textit{loves(Cat-I, Bird)}, where \textit{Cat-I} is assumed to be a unique instance of \textit{Cat}.\(^6\)

\(^5\)This is in keeping with the natural default meaning associated with statements like 'Cats prey on Birds', which generally means 'All Cats prey on all Birds'.

\(^6\)This is essentially the use of an skolem constant.
Note that this scheme interprets existential variables to be outside the scope of all the universally quantified variables.

For now let us assume that (missing details are provided below):

- Each type or instance is encoded as a $\rho$-btu node.
- Each conceptual IS-A relationship such as $\text{is-a}(A, B)$ is encoded using two connectionist links — a bottom-up link from $A$ to $B$ and a top-down link from $B$ to $A$, and
- The top-down and bottom-up links can be enabled selectively by built-in control mechanisms.

The time course of activation for the query: $\text{scared-of}(\text{Tweety}, \text{Sylvester})$? (Is Tweety scared of Sylvester?) is given in Fig. 2b. The query is posed by turning on $e:\text{scared-of}$ and activating the nodes $\text{Tweety}$ and $\text{Sylvester}$ in synchrony with the first and second arguments of $\text{scared-of}$, respectively. The bottom-up links emanating from $\text{Tweety}$ and $\text{Sylvester}$ are also enabled. The activation spreads along the conceptual hierarchy and eventually, $\text{Bird}$ and $\text{Cat}$ start firing in synchrony with $\text{Tweety}$ and $\text{Sylvester}$, respectively. At the same time, the activation propagates in the rule-base. Consequently, $c:\text{preys-on}$ becomes active and the first and second arguments of $\text{preys-on}$ begin firing in synchrony with the second and first arguments of $\text{scared-of}$, respectively. The net result is that the query $\text{scared-of}(\text{Tweety}, \text{Sylvester})$? is transformed into the query $\text{preys-on}(\text{Cat}, \text{Bird})$?. This query matches the stored fact $\text{preys-on}(\text{Cat}, \text{Bird})$ and leads to the activation of $c:\text{preys-on}$. In turn, $c:\text{scared-of}$ becomes active and signals an affirmative answer to the query.

4 Two technical problems

There are two technical problems that must be solved in order to integrate the conceptual hierarchy and the rule-based component.
4.1 Multiple Instantiation

The encoding of the IS-A hierarchy should be capable of representing multiple instantiations of a concept. For example, in the query discussed above, the concept Animal would receive activation originating at Tweety as well as Sylvester. 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.

4.2 Control of propagating activation

The encoding must provide built-in mechanisms for controlling the direction of activation in the IS-A hierarchy so as to correctly deal with queries containing existentially and universally quantified variables. Thus,

- Activation originating from an instance or a concept \( C \) that corresponds to a universally quantified variable in the query should propagate upwards to all its ancestors. By definition of universal quantification, if a fact is true for all objects of type \( C' \), \( C' \) being an ancestor of \( C \), then the fact is true for all subconcepts of \( C' \), including \( C \). Activation propagating upward is equivalent to checking if the relevant fact is universally true for some ancestor of \( C \), in which case it is true for all \( C \).

- 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 \( C \). A fact is true for some object of type \( C \) if at least one of the following holds:
  - The fact is true for all objects of type \( C' \), where \( C' \) is an ancestor of \( C \). Activation traveling upward from \( C \) checks this case.
  - The fact is true for some object of type \( C'' \), which is a descendent of \( C \). Activation traveling downward from \( C \) is meant to check this condition.

However, if the IS-A hierarchy permits multiple inheritance, then the fact would be true of \( C \) if it is true for all ancestors of a descendent of \( C \). This requires that activation must also propagate to the ancestors of the descendents of \( C \). The multiple inheritance situation is illustrated by the following fragment of a type hierarchy:

\[
\text{is-a}(\text{Bird, Animal}) \\
\text{is-a}(\text{Bird, Pet}) \\
\text{is-a}(\text{Canary, Bird}).
\]

If 'all pets are lovable', then it also follows that 'there exists some animal that is lovable'. Given the fact \( \forall x: \text{Pet} \text{lovable}(x) \) ('all pets are lovable'), to be able to give an affirmative answer to the query \( \exists x: \text{Animal lovable}(x) \) ? ('is there some animal that is lovable?'), we need to be able to propagate activation to all the ancestors of the descendents of Animal. The situation is illustrated in Fig. 3.

Thus, we require activation originating from a concept \( C \), which corresponds an existentially quantified variable, to propagate to its ancestors, descendents and ancestors of descendents.

The following section describes a solution to the above problems. The proposed solutions do not require an external controller to monitor and control the state of nodes in the network during the reasoning process.

5 Implementing the Type Hierarchy

5.1 Representing Entities

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. 4a. The entity cluster for \( C \) has \( k \) banks of \( \rho \)-btu nodes,
where $k$, the *multiple instantiation constant*, refers to the number of dynamic instantiations a concept can accommodate. Each bank $C_B$, consists of three $p$-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 $p$, then, $C_i$ fires in phase $p$. 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. 4a, $C_i$ 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 [6].

5.2 Switches

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. These specialized hardware structures interact with the units in the entity cluster to select a maximum of $k$ phases, $p_1, \ldots, p_k$, such that the activations in these phases are channeled to $C_1, \ldots, C_k$. 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, $1 \leq i \leq k$. The bottom-up switch has $kn_{sub}$ inputs while the top-down switch has $kn_{sup}$ inputs, $n_{sub}$ and $n_{sup}$ being the number of sub- and super-concepts of $C$, respectively. Further, there is also a feedback from the $C_i$ nodes to both the switches (See Fig. 4a and Fig. 5.)

The interaction between the switches and the entity cluster (Fig. 4a) brings about efficient and automatic dynamic allocation of banks in an entity cluster, by ensuring the following characteristics:

- 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. 5. The $k$ $p$-btu nodes, $S_1, \ldots, S_k$, with their associated $\tau$-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 $S_2, \ldots, S_k$; these inputs directly connect to $S_1$. As a result of these excitatory-inhibitory connections, the $S_2, \ldots, S_k$ 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 inputs to the switch directly connect to $S_1$ (Fig. 5). In keeping with the behavior of \textit{p-btu} nodes (Section 2), $S_1$ becomes active in response to the first available input and continues to fire in phase with that input as long the input remains active. As $S_1$ goes active, the $\tau$-or node associated with $S_1$ turns, thereby enabling $S_2$ (via the link). Inhibitory feedback from $C_1$ (via the link) ensures that $S_2$ is \textit{not} enabled during the phase $\rho$ in which $C_1$ is firing. Thus $S_2$ selects and starts firing in a phase \textit{other than} $\rho$. Once $S_2$ has made its selection, $S_3$ gets its turn, and so on.

Note that, in general, $C_i$ could receive input in \textit{two} phases— one from the bottom-up switch for $C_i$, and another from its top-down switch. $C_i$ being a \textit{p-btu} node, picks one of these phases to fire in. As instantiations are deputed to the entity cluster, the \textit{p-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 $\rho \not\in \{\rho_1, \ldots, \rho_{i-1}\}$, 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 \textit{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 mechanism 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.

5.3 Connecting up the Type Hierarchy

A fact of the form $is-a(A,B)$ is represented as shown in Fig. 4b by:

- connecting the $A_{iτ}$, $i = 1, \ldots, k$ nodes to the bottom-up switch for $B$;
- connecting the $B_{i↓}$, $i = 1, \ldots, k$ 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 $C_{iτ}$ node is now receiving two inputs in phase $ρ$ (from the bottom-up switch and from $C_i$; see Fig. 4a.) Since it has a threshold $θ = 2$, $C_{iτ}$ also fires in phase $ρ$. This causes activation in phase $ρ$ to eventually 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 (Section 4.2). Similarly, when $C_i$ receives activation from the top-down switch in phase $ρ$, both $C_i$ and $C_{i↓}$ become active in phase $ρ$. $C_{i↓}$ follows suit, because of the link from $C_{i↓}$ to $C_{iτ}$, so that the whole bank $C_B$, 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, which is in keeping with the desired behavior mentioned in Section 4.2.

6 Combining the Type Hierarchy with the Rule-Based Reasoner

6.1 Typed Variables in Facts

As mentioned in Section 3,
A typed, universally quantified variable is treated as being equivalent to its type, and vice versa. Thus the facts $\forall x:C_A\, y:C_B\, P(x,y)$, $\forall x:C_A\, P(x,C_B)$ and $\forall y:C_B\, P(C_A,y)$ are equivalent to $P(C_A,C_B)$. A typed, existentially quantified variable is encoded by creating a unique subconcept of the associated type. Thus, $\exists x:C_A\, P(x,C_B)$ is encoded as $P(C_i,C_B)$, where $C_i$ is a unique subconcept of $C_A$.

This interpretation forces all existential variables to be outside the scope of the universal variables in the fact. Further, as noted in Section 2, any untyped variable in a fact is treated as being existentially quantified (unspecified role). For example $\forall x:Cat\, preys-on(x,y)$ would be interpreted as ‘Every cat preys on some bird’.

Concepts and instances can now accommodate $k$ instantiations. A fact or rule encoding should therefore be able to check if any one of these $k$ instantiations is in the required phase. For example, the fact $P(C_A)$ should be recognized if the argument of predicate $P$ fires in synchrony with some $C_A$, $1 \leq i \leq k$. Similarly, the rule $\forall x, y\, P(x,y) \Rightarrow Q(x,y,C_A)$ should fire if the third argument of $Q$ is in synchrony with any of the $C_A$ nodes. This effect is easily realized by treating the output of $C_A$ to be a bundle of $k$ links. Fig. 6 shows the encoding of the fact $P(C_A,C_B)$, where the outputs of $C_A$ and $C_B$ are considered to be bundles of $k = 3$ wires. Fig. 7 illustrates rule-encoding in the light of the fact that entities are clusters of $k$ nodes. A comparison with [8] (also see Fig. 1a) indicates that a link from a constant is now replaced by a bundle of $k$ links from the corresponding concept or instance.

### 6.2 Typed Variables in Queries

Consider a query $P(\ldots, z, \ldots)$? where the $i$-th argument of predicate $P$ is filled by the typed variable $z$. Let $C_A$ be the type of $z$. Depending on whether $z$ is universally or existentially quantified (Section 4.2), the query is posed as follows (after turning on the enabler $e:P$ of predicate $P$):

- **If $z$ is universally quantified,** (i.e., the query is of the form $\forall x:C_A\, P(\ldots, z, \ldots)$?), then $C_{A1}$ and $C_{A11}$ are set to fire in synchrony with the $i$-th argument of $P$.

  In the type hierarchy, the universal typed variable $z$ sets off an upward trail of activation originating at $C_A$, the type of $z$. The justification for this is as follows: if some fact is asserted for all objects of type $C$, then the fact is true for all the sub-concepts or descendents of $C$. Thus, if we need to check if $P(\ldots, C_A, \ldots)$ is true, we need to see if $P(\ldots, C, \ldots)$ is asserted in the system where $C$ is either $C_A$ or an ancestor of $C_A$. Turning $C_{A1}$ and $C_{A11}$ exactly achieves this goal, as mentioned in Section 4.2.

- **If $z$ is existentially quantified,** (i.e., the query is of the form $\exists x:C_A\, P(\ldots, z, \ldots)$?), then $C_{A1}$ and $C_{A11}$ are set to fire in synchrony with the $i$-th argument of $P$.

\[7\text{Assume backward reasoning.}\]
This causes activation to spread to the ancestors, descendents and the ancestors of the descendents of $C_A$, the type of $x$. The justification for this kind of complex activation propagation is the following: a fact involving $P$ is true for some object of type $C_A$, if at least one of the following holds:

- The fact is true for all objects of type $C'$, where $C'$ is an ancestor of $C_A$; in this case the fact is true for all objects of type $C_A$ and is therefore true for some object of type $C_A$. This condition is checked by activation traveling upward from $C_A$.

- The fact is true for some object of type $C''$, where $C''$ is a descendent of $C_A$. This case is handled by activation traveling downward from $C_A$.

- The fact is true for all objects of type $C'''$, where $C'''$ is an ancestor of some descendent, $C$, of $C_A$. In such a situation, the fact holds for all objects of type $C$ and hence for some object of type $C_A$. Activation traveling up from every descendent of $C_A$ is meant to catch this situation.

In the context of the type hierarchy, the above cases are the only ones which assert $\exists z: C_A\ P(z, r, \ldots)$? No other cases are possible, reiterating what was stated in Section 4.2.

Just as for facts, concepts directly specified in the query predicate are a shorthand for a universal typed variable (i.e., $P(\ldots, C_A, \ldots)$? is the same as $\forall z: C_A\ P(z, r, \ldots)$?). Universally quantified variables are interpreted to be within the scope of the existentially quantified variables. Untyped variables are unspecified roles, and hence will not be assigned a phase.

Example

Assuming that the concepts in the type hierarchy shown in the network of Fig. 2a had the structure indicated in Fig. 4a, the query $\exists z: Cat\ loves(z, Tweety)$? would be posed by:

- Turning $Cat_1$ and $Cat_{11}$ to fire in sync with lover;
- Turning $Tweety_1$ and $Tweety_{11}$ to fire in sync with lovee;
- Activating the enabler of the loves predicate.
Figure 8: Trace of spreading activation for the query $\exists x: \text{Cat loves}(x, \text{Tweety})$?. The suffixes up and down represent $\uparrow$ and $\downarrow$ respectively. Activation of only those nodes relevant to the example are shown.

Since $\text{Cat}_1$ is associated with an existential variable, activation from $\text{Cat}_1$ spreads to $\text{Animal}_1$ (upward) and to $\text{Cat}-1_1$ (downward). Also, activation from $\text{Cat}_1$ spreads to the ancestors of its descendants. Since the ancestors of the descendents of $\text{Cat}_1$ are $\text{Cat}_1$ and $\text{Animal}_1$, and since they are already firing in phase with $\text{Cat}_1$, no new instantiations are introduced. Activation from $\text{Tweety}_1$ propagates only upward, to $\text{Canary}_1$, $\text{Bird}_1$ and $\text{Animal}_2$, since it is associated with a universal variable. Fig. 8 shows the resulting spread of activation in the network. The activity of the corresponding $\uparrow$ and $\downarrow$ nodes are also indicated in Fig. 8.

Activation spreading downward from $\text{Cat}_1$ turns $\text{Cat}-1_1$, while upward activation from $\text{Tweety}_1$ eventually reaches $\text{Bird}_1$. When this happens, the fact node $F2$ corresponding to the fact $\exists x: \text{Cat}, \forall y: \text{Bird loves}(x, y)$ turns result in an affirmative answer to the query. Other queries are handled similarly.

### 6.3 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 suitably modify the implementation of rules: In a forward reasoning system, the rule is encoded by introducing a $\tau$-or node to perform the type checking for the argument under question. Fig. 9a, shows the encoding of the rule $\forall x:T_1, \forall :T_2 \exists :T_3 P_1(x, y) \& P_2(x, z) \Rightarrow Q(y)$. Here, $g_1$, $g_2$ and $g_3$ are $\tau$-or nodes for type checking. These nodes turn only if the corresponding predicate arguments are bound to objects of the right type. For example, $g_3$ would go 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 $T_3$. Note that $g_1$ also checks if the first arguments of both $P_1$ and $P_2$ are in the phase of $T_1$. Thus $g_1$ enforces type restrictions and checks for repeated variables. The links from $T_1, T_2$ and $T_3$ are actually bundles of $k$ wires each (as indicated in Fig. 7b). It is also evident from the Fig. 9a that the rule will not fire (and $Q$ will not go active) unless all the $g$-nodes ($g_1$, $g_2$ and $g_3$) are so on. In the forward reasoner, typed variables are allowed only in the antecedent of the rule.
Figure 9: (a) 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. (b) Network encoding the rule $\forall x:T_1 \exists y:T_2 P(x) \Rightarrow Q(x,y)$ in a backward reasoning system. $T'_2$ is a unique subconcept of $T_2$. Again, links from $T_1$ and $T'_2$ are bundles of $k$ wires.

In a backward reasoner, the strategy is similar, except that:

- Type checking for a typed universally quantified variable is enforced by an inhibitory link from the concept representing the type of the concerned argument.
- For a typed, existentially quantified variable, the inhibitory links for type enforcement are derived from a unique subconcept of the associated type.

The network which implements the rule $\forall x:T_1 \exists y:T_2 P(x) \Rightarrow Q(x,y)$ for backward reasoning is sketched in Fig. 9b. To encode the typed existentially quantified variable $y$, inhibitory links are derived from $T'_2$, which is a unique subconcept of $T_2$, the type of $y$. A type mismatch causes $g_1$ to block further propagation of activation. In the backward reasoner, typed variables are allowed only in the consequent.

Both in the forward and backward reasoners, as is evident from the paragraphs above, a rule fires only if all typed arguments are firing in synchrony with their respective types.

7 Conclusions

Adding a type hierarchy allows the connectionist reasoning system to handle rules, facts, and queries with typed variables. This expands the power of the rule-based reasoner by making the reasoner cope with a much wider set of rules and facts. The system described here has been simulated using the simulator described in [5], which is an extension of the Rochester Connectionist Simulator [3].

Several extensions to the system proposed here are being investigated:

---

8Actually, a bundle of $k$ inhibitory links, one from each of the $k$ nodes in the cluster for the concept. See Fig. 7a.

9This is similar to the manner in which typed existential variables in a fact are interpreted.
• 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 in which the quantifiers can occur in any arbitrary order.

• We are also working on the design of an expanded system that would allow property-value attachments to concepts, which would support inheritance and recognition [6].

• We also wish to combine the IS-A hierarchy with a reasoning system that allows multiple instantiation of predicates and combines the forward and backward reasoners to make use of both long-term and dynamic (temporary) facts during reasoning. Such a system already exists except that multiple instantiation in the forward reasoner is very inefficient in the use of links. Attempts are being made to improve the situation.

References


