May 1991

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

Lokendra Shastri
University of Pennsylvania

D. R. Mani
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-91-33.

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

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-91-33.

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 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.

*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, $\forall x{:}Cat,\ y{:}Bird\ preys\text{-}on(x,y)$ and $\exists x{:}Cat,\ \forall y{:}Bird\ loves(x,y)$ 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 $\exists x{:}Cat\ \forall y{:}Bird\ scared\text{-}of(y,x)?$, 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:

  $\forall x{:}animate,\ y{:}solid\text{-}obj\ walk\text{-}into(x,y) \Rightarrow hurt(x)$
  $\forall x{:}animate\ \exists y{:}animate\ predator(x) \Rightarrow preys\text{-}on(x,y)$

  The first rule is applicable only if the two arguments of 'walk-into' are of the type 'animate' and 'solid-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*:

[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 Fahlman'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'.

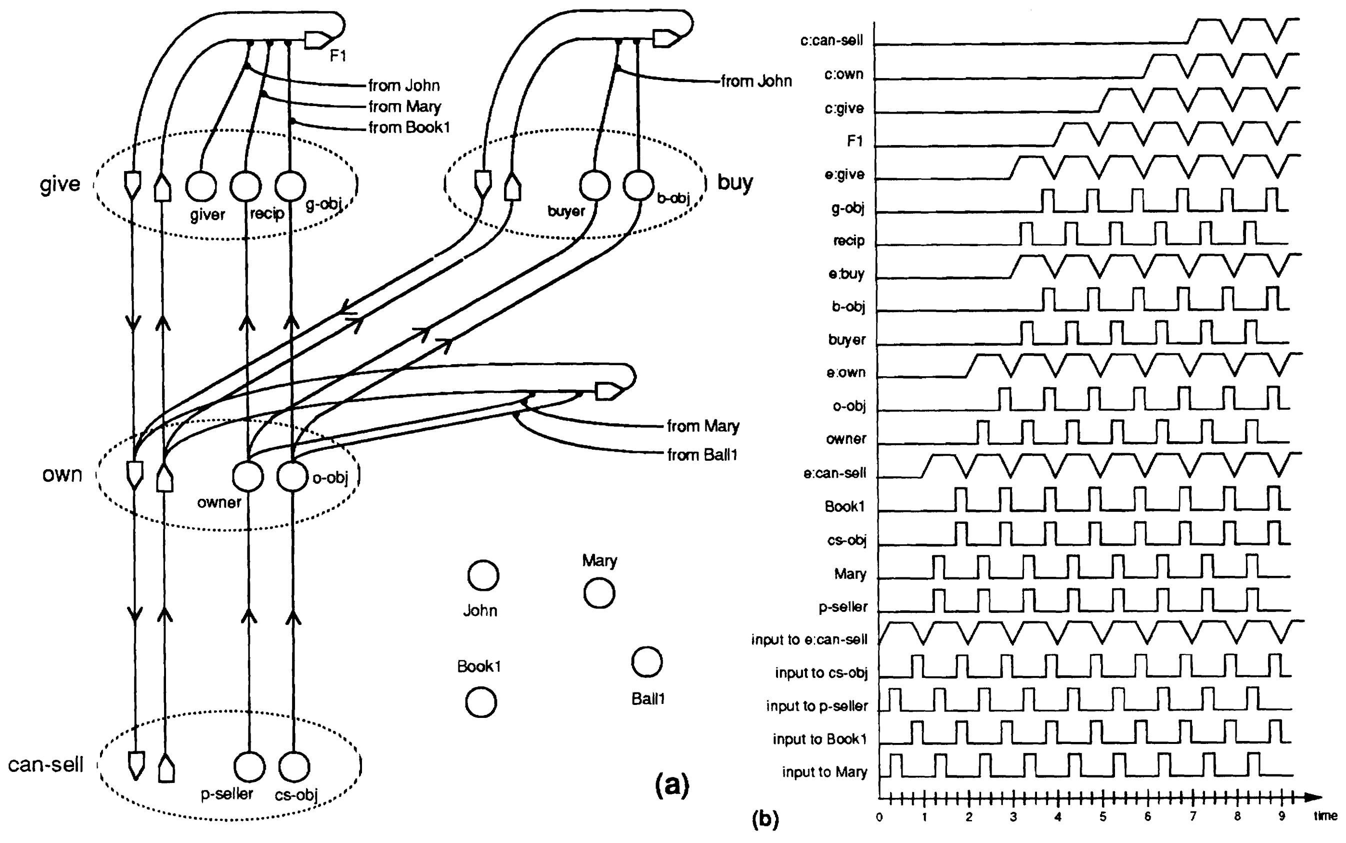


Figure 1: (a) An example encoding of rules and facts. (b) Activation trace for the query *can-sell(Mary,Book1)?*.

$\forall x, y, z$ *give(x,y,z)* $\Rightarrow$ *own(y,z)*
$\forall x, y$ *buy(x,y)* $\Rightarrow$ *own(x,y)*
$\forall x, y$ *own(x,y)* $\Rightarrow$ *can-sell(x,y)*
give(John,Mary,Book1)
buy(John,x)
own(Mary,Ball1).

The encoding makes use of two types of nodes. These are 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*. 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.

[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.

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[4] 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 modifier 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.)

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 π 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 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 ($1 \leq i \leq k$) 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 *can-sell(Mary,Book1)?* (i.e., Can Mary sell Book1?) This query is posed by providing inputs to the constants *Mary* and *Book1*, the arguments *p-seller*, *cs-obj* and the *enabler e:can-sell* as shown in Fig. 1b. *Mary* and *p-seller* receive in-phase activation and so do *Book1* and *cs-obj*. Let us refer to the phase of activation of *Mary* and *Book1* as phase-1 and phase-2 respectively. 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. 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 system has essentially, created dynamic bindings for the arguments of predicate *own*. *Mary* has been bound to the argument *owner*, and *Book1* has been bound to the argument *own-object*. These newly created bindings in conjunction with the activation of *e:own* can be thought of as encoding the query *own(Mary,Book1)?* (i.e., 'Does Mary own Book1?')! The τ-and node associated with the fact *own(Mary, Ball1)* does not match the query and remains inactive. The activations from *owner* and *o-obj* reach the arguments *recip* and *g-obj* of *give*, and *buyer* and *b-obj* of *buy* respectively. Thus beginning with the third period of oscillation, arguments *recip*

[4] We assume backward reasoning.

and *buyer* become active in phase-1, while arguments *g-obj* and *b-obj* become active in phase-2. In essence, the system has created new bindings for the predicates *can-sell* and *buy* that can be thought of as encoding two new queries: *give(x,Mary,Book1)?* (i.e., 'Did *someone* give Mary Book1?'), and *buy(Mary,Book1)?*. Observe that now the τ-and node associated with the fact *give(John,Mary,Book1)* (this is the τ-and node labeled F1 in Fig. 1a), becomes active as a result of the uninterrupted activation from *e:give*. The inhibitory inputs from *recip* and *g-obj* are blocked by the in-phase inputs from *Mary* and *Book1*, respectively. The activation from this τ-and node 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 resulting in an affirmative answer to the query *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(y,x)*.

The former fact 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(Chirpy,Robin)
is-a(Tweety,Canary)
is-a(Sylvester,Cat).

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 *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 assumed to be a unique instance of *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.

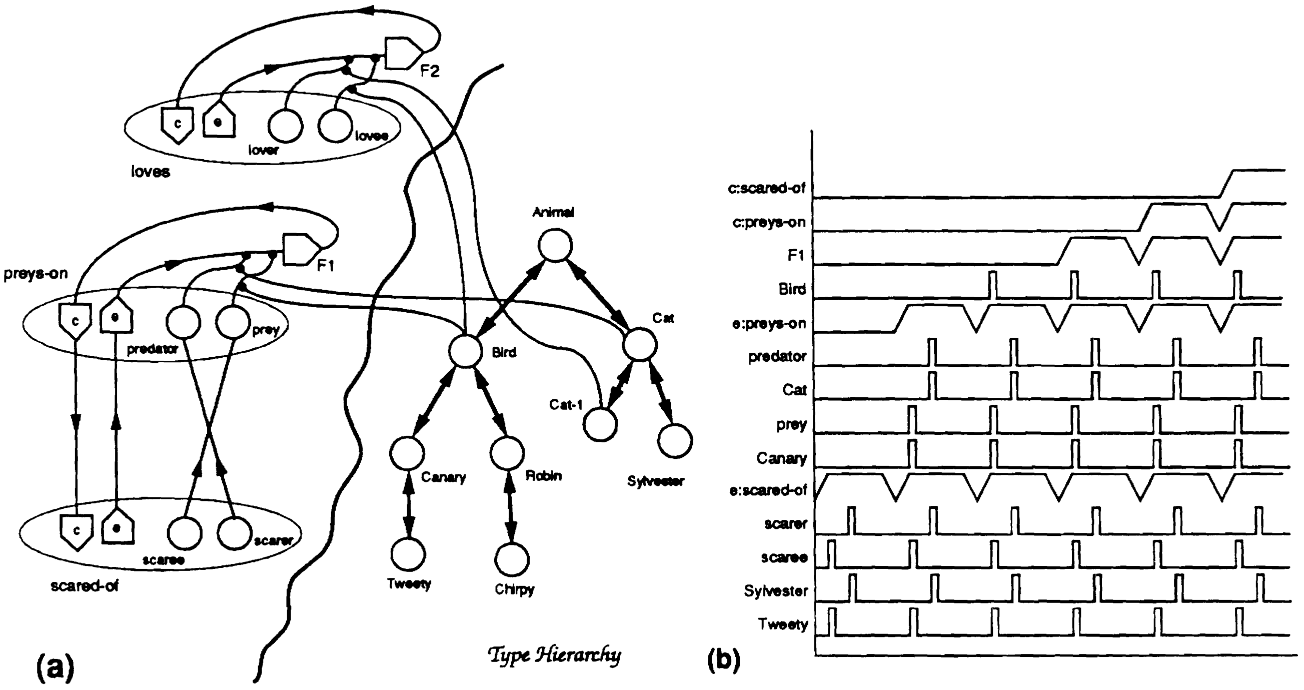


Figure 2: (a) An example network. (b) Trace of spreading activation for the query $\exists x$*:Canary scared-of(x,Sylvester)?*.

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 ρ-btu node.
- 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
- The top-down and bottom-up links can be enabled selectively by built-in control mechanisms.

The time course of activation for the query: *scared-of(Tweety,Sylvester)?* (Is Tweety scared of 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 activation spreads along the conceptual hierarchy and eventually, *Bird* and *Cat* start firing in synchrony with *Tweety* and *Sylvester*, respectively. At the same time, the activation propagates in the rule-base. Consequently, *e:preys-on* becomes active and the first and second arguments of *preys-on* begin firing in synchrony with the second and first arguments of *scared-of*, respectively. The net result is that the query *scared-of(Tweety,Sylvester)?* is transformed into the query *preys-on(Cat,Bird)?*. This 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.

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:

  is-a(Bird,Animal)
  is-a(Bird,Pet)
  is-a(Canary,Bird).

  If 'all pets are lovable', then it also follows that 'there exists some animal that is lovable'. Given the fact $\forall$*x:Pet lovable(x)* ('all pets are lovable)', to be able to give an affirmative answer to the query $\exists$*x: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 ρ-btu nodes,

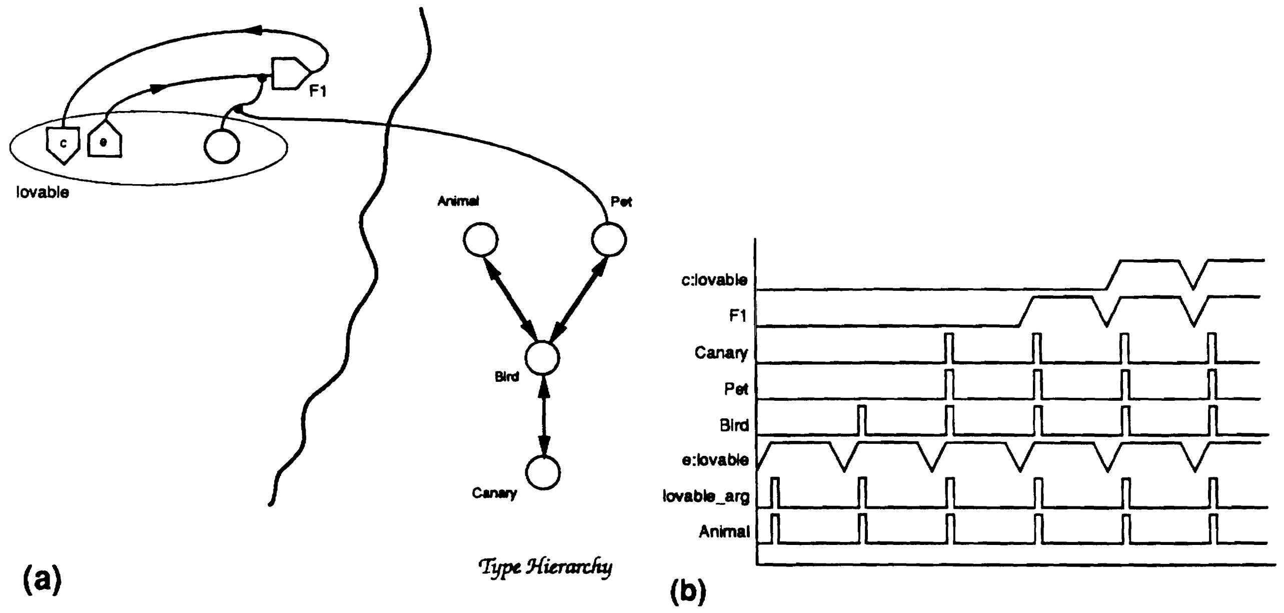


Figure 3: (a) A fragment of a type hierarchy with multiple inheritance, encoding the fact $\forall x$:*Pet lovable(x)*. (b) Activation trace for the query $\exists x$:*Animal lovable(x)?*.

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. 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].

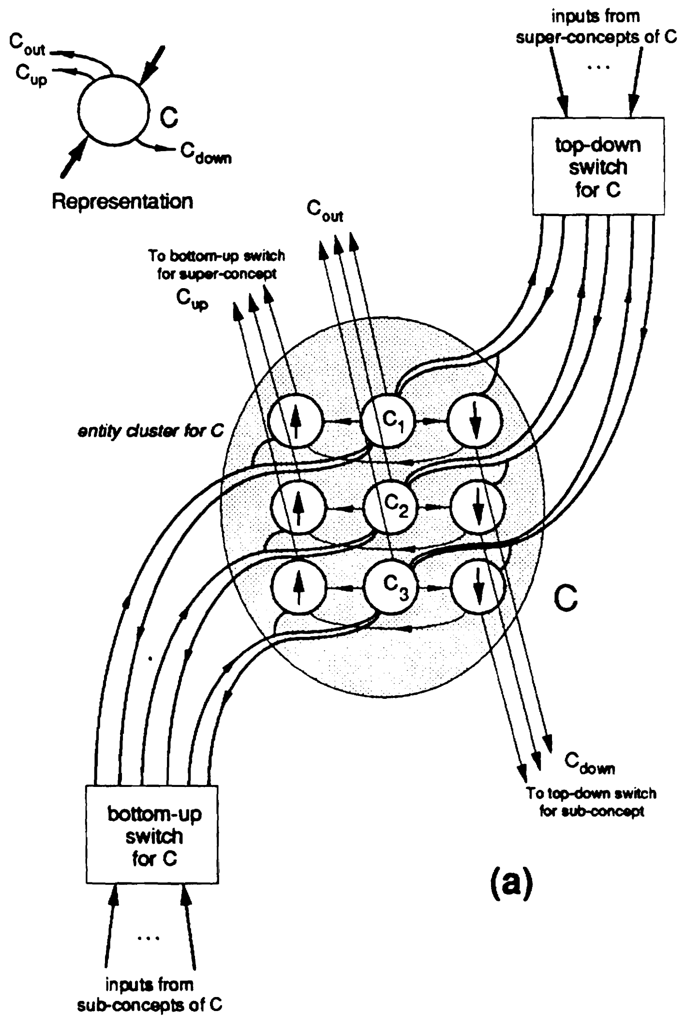
5.2 Switches

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, $\rho_1, \ldots, \rho_k$, 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 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



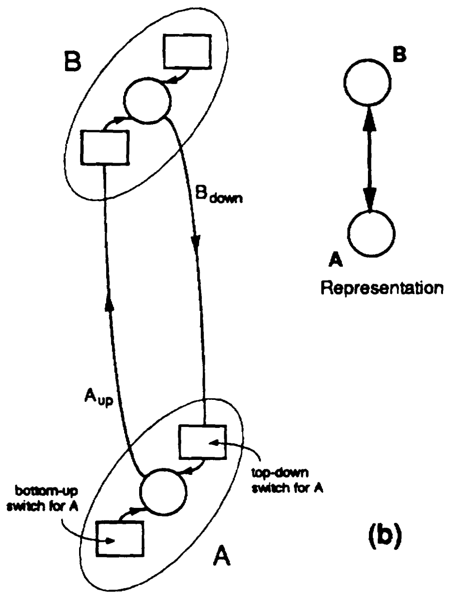


Figure 4: (**a**) Structure of the entity cluster for C, and its interaction with the bottom-up and top-down switches. The $\uparrow$ and $\downarrow$ nodes have a threshold $\theta = 2$. The multiple instantiation constant, $k = 3$. (**b**) Encoding of the *is-a* relation *is-a(A,B)*. A bundle of k wires is represented by a single link.

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 ρ-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 τ-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 *not* enabled during the phase ρ in which C_1 is firing. 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.

Note that, 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. 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

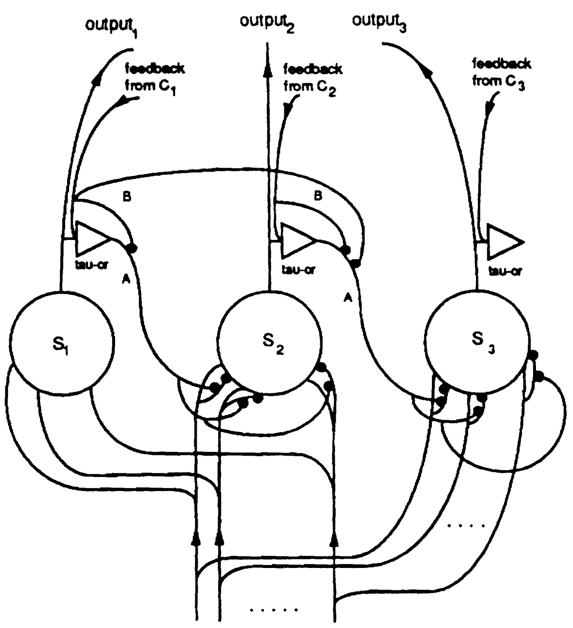


Figure 5: Architecture of the switch. The multiple instantiation constant $k = 3$.

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 atmost 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 $A_{i\uparrow}, i = 1, \ldots, k$ nodes to the bottom-up switch for B;
- 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 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 $\theta = 2$, $C_{i\uparrow}$ 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 $C_{i\downarrow}$ 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, 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,

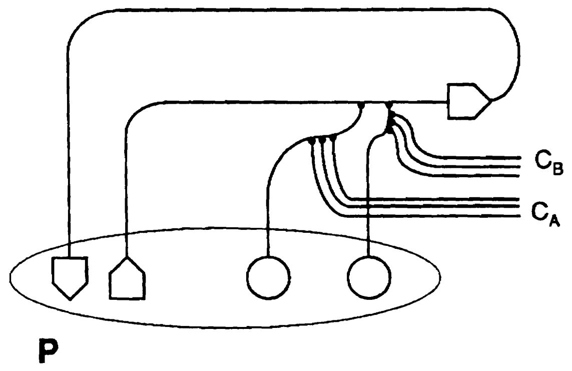


Figure 6: Encoding of the fact $P(C_A,C_B)$. Multiple instantiation constant $k = 3$.

- 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 $P(C'_A,C_B)$, where C'_A 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*** $C_{Ai}, 1 \leq i \leq k$. Similarly, the rule[7] $\forall x,y\ P(x,y) \Rightarrow Q(x,y,C_A)$ should fire if the third argument of Q is in synchrony with ***any*** of the C_{Ai} 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 $P(\ldots,x,\ldots)?$ where the i-th argument of predicate P is filled by the typed variable x. Let C_A be the type of x. Depending on whether x is universally of existentially quantified (Section 4.2), the query is posed as follows (after turning on the enabler $e{:}P$ of predicate P):

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

  In the type hierarchy, the universal typed variable x sets off an ***upward trail*** of activation originating at C_A, the type of x. 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 $C_{A1\uparrow}$ exactly achieves this goal, as mentioned in Section 4.2.

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

[7] Assume backward reasoning.

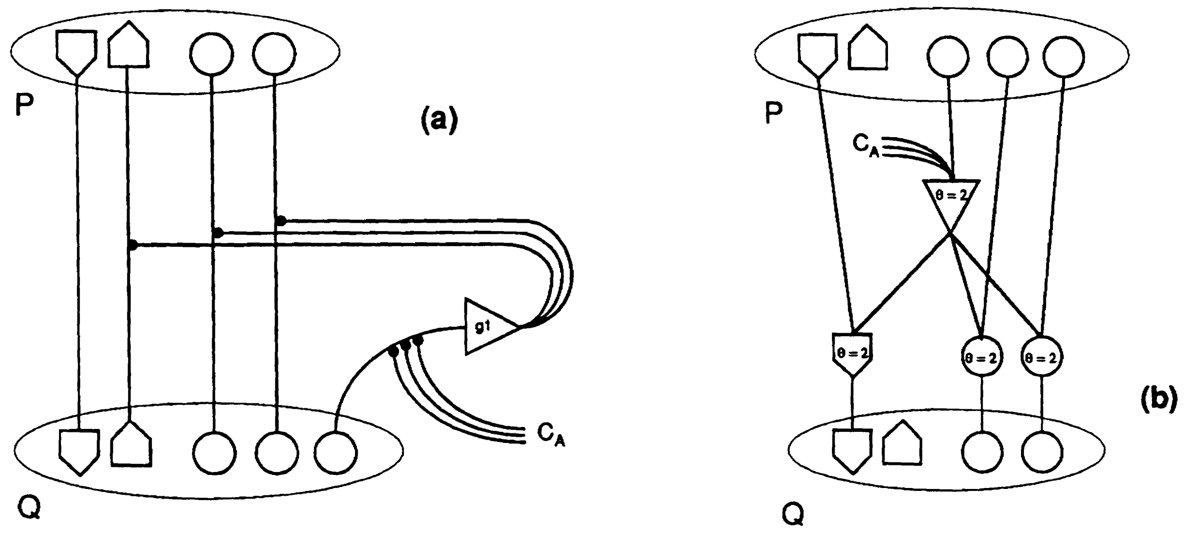


Figure 7: (a) Encoding of the rule $\forall x,y$ *P(x,y)* $\Rightarrow$ *Q(x,y,C_A)* for a *backward reasoning* system. (b) Encoding of the rule $\forall x.y$ *P(C_A,x,y)* $\Rightarrow$ *Q(x,y)* for a *forward reasoning* system.

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 $\exists x{:}C_A$ *P(...,x,...)?* 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 $\forall x{:}C_A$ *P(...,x,...)?*). 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 $\exists x{:}Cat$ *loves(x,Tweety)?* would be posed by:

- Turning Cat_{l} and $Cat_{\mathrm{l}\downarrow}$ to fire in sync with *lover*;
- Turning $Tweety_{\mathrm{l}}$ and $Tweety_{\mathrm{l}\uparrow}$ to fire in sync with *lovee*;
- Activating the enabler of the *loves* predicate.

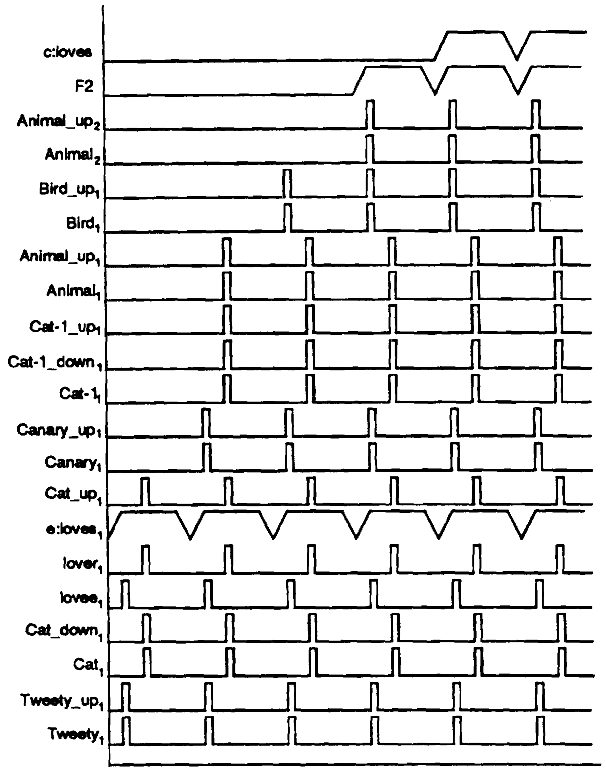


Figure 8: Trace of spreading activation for the query $\exists x$*:Cat loves(x, Tweety)?*. The suffixes **up** and **down** represent $\uparrow$ and $\downarrow$ respectively. Activation of only those nodes relevant to the example are shown.

Since Cat_1 is associated with an existential variable, activation from Cat_1 spreads to $Animal_1$ (upward) and to $Cat\text{-}1_1$ (downward). Also, activation from Cat_1 spreads to the ancestors of its descendants. Since the ancestors of the descendents of Cat_1 are Cat_1 and $Animal_1$, and since they are already firing in phase with Cat_1, no new instantiations are introduced. Activation from $Tweety_1$ propagates only upward, to $Canary_1$, $Bird_1$ and $Animal_2$, 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 Cat_1 turns $Cat\text{-}1_1$, while upward activation from $Tweety_1$ eventually reaches $Bird_1$. When this happens, the fact node $F2$ corresponding to the fact $\exists x$*:Cat,* $\forall y$*:Bird loves(x,y)* turns .resulting 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 τ-or node to perform the type checking for the argument under question. Fig. 9a, shows the encoding of the rule $\forall x{:}T_1,\ y{:}T_2\ \exists z{:}T_3\ P_1(x,y)$ & $P_2(x,z) \Rightarrow Q(y)$. Here, g_1, g_2 and g_3 are τ-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 sc on. In the forward reasoner, *typed variables are allowed only in the antecedent of the rule.*

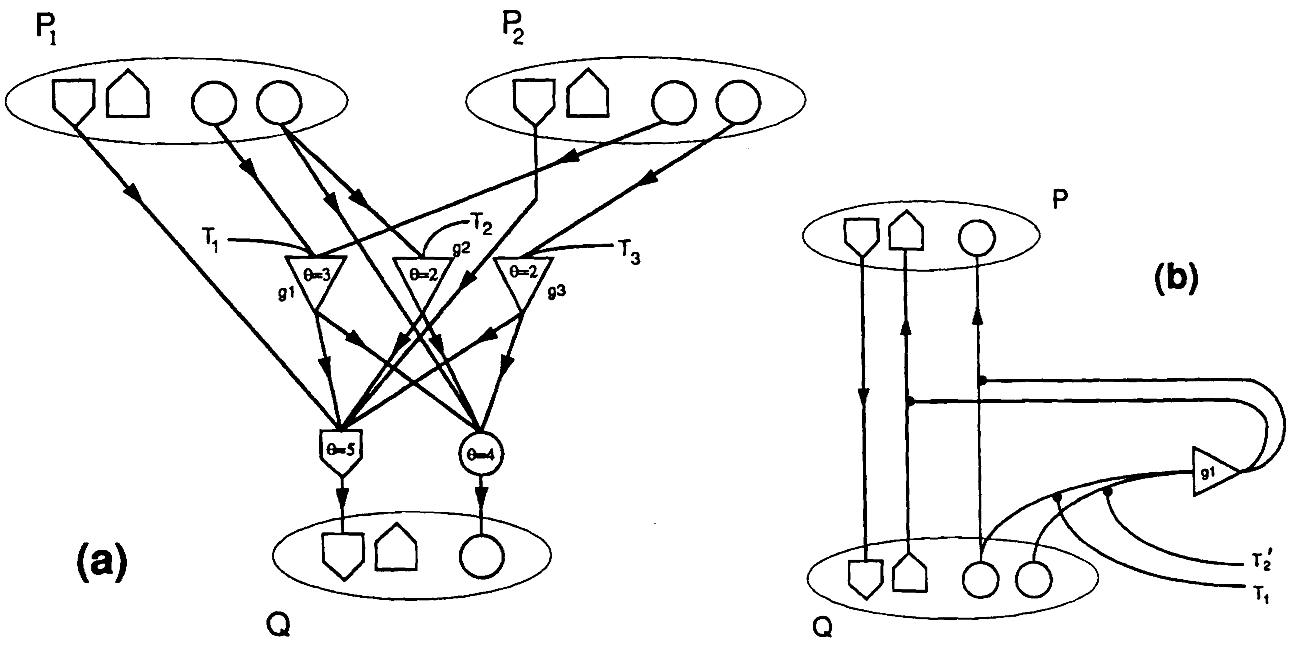


Figure 9: (a) Network encoding the rule $\forall x{:}T_1,\ y{:}T_2\ \exists z{:}T_3\ P_1(x,y)\ \&\ P_2(x,z) \Rightarrow Q(y)$ in a forward reasoning system. Links from T_1, T_2 and T_3 are actually bundles of k wires carrying the k instantiations of these constants. (b) Network encoding the rule $\forall x{:}T_1\ \exists y{:}T_2\ P(x) \Rightarrow Q(x,y)$ 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[8] 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[9].

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 parasgraphs 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:

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

[1] Ajjanagadde, V. and Shastri, L. Rules and variables in neural nets, *Neural Computation.* To appear in Vol. 3, No. 1, 1991.

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

[3] Goddard, N. H., Lynne, K. J., Mintz, T. and Bukys, L. *Rochester Connectionist Simulator.* Technical Report 233 (revised), University of Rochester, October 1989.

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

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

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

[7] Shastri, L. and Ajjanagadde, V. An optimally efficient limited inference system. *Proceedings of AAAI-90, the Twelfth National Conference of the American Association of Artificial Intelligence*, Cambridge MA, July 1990, pp. 563–570.

[8] Shastri, L. and Ajjanagadde, V. *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, University of Pennsylvania, Jan 1990.

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