January 1992

Multiple Instantiation of Predicates in a Connectionist Rule-Based Reasoner

D. R. Mani
University of Pennsylvania

Lokendra Shastri
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No.MS-CIS-92-05.

Multiple Instantiation of Predicates in a Connectionist Rule-Based Reasoner

Abstract

Shastri and Ajjanagadde have described a neurally plausible system for knowledge representation and reasoning that can represent systematic knowledge involving *n*-ary predicates and variables, and perform a broad class of reasoning with extreme efficiency. The system maintains and propagates variable bindings using temporally synchronous — i.e., in-phase — firing of appropriate nodes. This paper extends the reasoning system to incorporate *multiple instantiation of predicates*, so that any predicate can now be instantiated with up to *k* dynamic facts, *k* being a system constant. The ability to accommodate multiple instantiations of a predicate allows the system to handle a much broader class of rules; the system can even handle limited recursion (up to *k* levels). Though the time and space requirements increase by a constant factor, the extended system can still answer queries in time proportional to the length of the shortest derivation of the query and is independent of the size of the knowledge base.

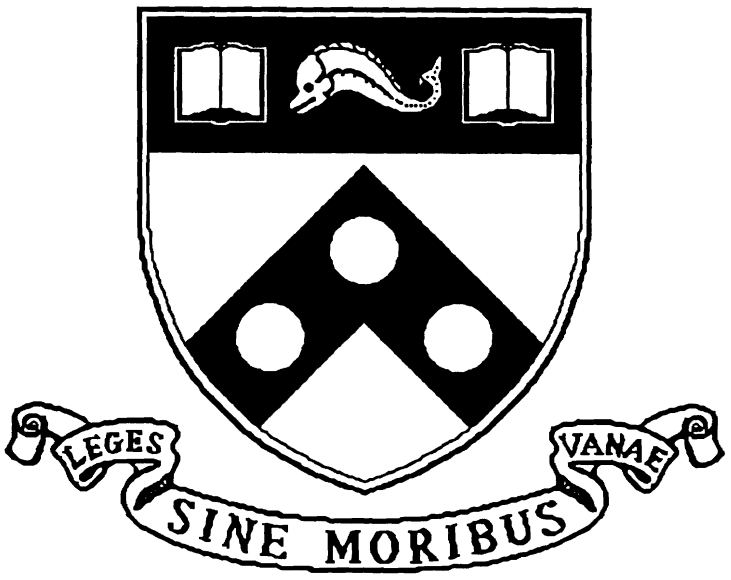
Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No.MS-CIS-92-05.

Multiple Instantiation Of Predicates In A Connectionist Rule-Based Reasoner

MS-CIS-92-05
LINC LAB 212

D.R. Mani
Lokendra Shastri



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

January 1992

Multiple Instantiation of Predicates in a Connectionist Rule-Based Reasoner*

D. R. Mani and **Lokendra Shastri**

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104, USA
mani@linc.cis.upenn.edu

January 1992

Abstract

Shastri and Ajjanagadde have described a neurally plausible system for knowledge representation and reasoning that can represent systematic knowledge involving n-ary predicates and variables, and perform a broad class of reasoning with extreme efficiency. The system maintains and propagates variable bindings using temporally synchronous — i.e., in-phase — firing of appropriate nodes. This paper extends the reasoning system to incorporate *multiple instantiation of predicates*, so that any predicate can now be instantiated with up to k dynamic facts, k being a system constant. The ability to accommodate multiple instantiations of a predicate allows the system to handle a much broader class of rules; the system can even handle limited recursion (up to k levels). Though the time and space requirements increase by a constant factor, the extended system can still answer queries in time proportional to the length of the shortest derivation of the query and is independent of the size of the knowledge base.

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

1 Introduction

In [Shastri & Ajjanagadde 1990b, Shastri & Ajjanagadde 1990a, Ajjanagadde & Shastri 1991], Shastri and Ajjanagadde have described a solution to the variable binding problem ([Feldman 1982, Malsburg 1986]) 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 extended to incorporate multiple instantiation of predicates. With this ability, the system can *simultaneously* represent multiple dynamic facts about a predicate. For example, the dynamic facts *loves(John,Mary)* and *loves(Mary,Tom)* can now be represented *at the same time*. As a result, the extended system can represent and reason using a set of rules which cause a predicate to be instantiated more than once; We can use rules of the form 'if x is a sibling of y, then y is a sibling of x' where we represent the symmetric nature of the sibling relation; In a similar manner, we can also represent (limited) transitivity of a predicate. Section 3 explores these advantages in detail.

Section 2 provides a brief overview of the rule-based reasoning system. Section 3 provides an overview of the multiple instantiation reasoning system and Sections 4 and 5 discuss its implementation in detail. Most of the time, we will concern ourselves only with backward reasoning. Forward reasoning will be considered only briefly in Section 6. The text will therefore, by default, refer to backward reasoning unless explicitly stated otherwise. We close with some concluding remarks.

2 The rule-based reasoning system

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

$\forall x, y, z$ *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

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

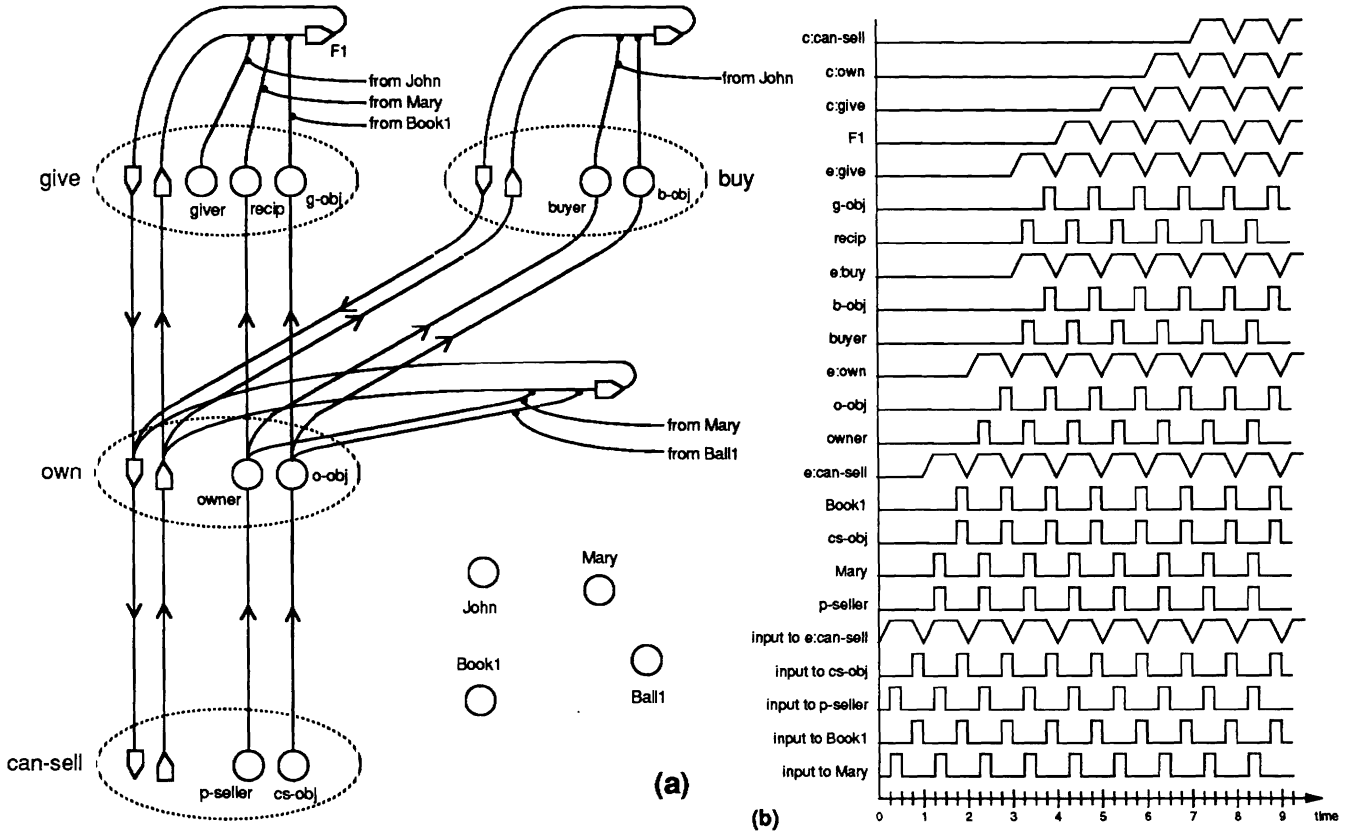


Figure 1: (a) An example encoding of rules and facts. (b) Activation trace for the query *can-sell(Mary,Book1)?*.

of the pulses produced by a τ-and node equals π.[2] 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 Figure 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[3] 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

[2]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.

[3]We assume backward reasoning.

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 Figure 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 Figure 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* 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 Figure 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 Figure 1b).

3 Multiple Instantiation – An Overview

As mentioned in Section 1, being able to represent multiple dynamic facts about the same predicate provides several additional capabilities not possible in the original reasoner. Introduction of multiple instantiation relies on the assumption that, during an episode of reflexive reasoning, any given predicate need only be

instantiated a *bounded* number of times. In [Shastri & Ajjanagadde 1990b], it is argued that a reasonable value for this bound is around three to five. We shall refer to this bound as the *multiple instantiation constant*, k.

The extended reasoning system can represent multiple dynamic facts (or *instantiations*) about the *same* predicate. For example, we can have dynamic activation representing *loves(John,Mary)*, *loves(Mary,Tom)* and *loves(Tom,Susan) simultaneously.* As a consequence of being able to do this, we can represent rules like

$$\forall x,y,z\ \exists t\ move(x,y,z) \Rightarrow present(x,y,t)$$
$$\forall x,y,z\ \exists t\ move(x,y,z) \Rightarrow present(x,z,t)$$

where the first rule states 'if x moves from y to z, then there is a time when x was present at y', and the second rule states 'if x moves from y to z, then there is a time when x was present at z'. These two rules are equivalent to stating 'if x moves from y to z, then there is a time when x is present at y *and* there is a time when x is present at z:

$$\forall x,y,z\ \exists t_1,t_2\ move(x,y,z) \Rightarrow present(x,y,t_1) \wedge present(x,z,t_2).$$

On encoding the above rule in a backward reasoning system, a query like *present(John,New-York,t)?* would result in *two* instantiations of *move*: *move(John,New-York,z)* and *move(John,y,New-York)*. *Either* of these dynamic instantiations may match a long-term *move* fact to provide an affirmative answer to the posed query.

The system can also represent rules with a *cyclic* predicate connection graph. A simple example of such a rule is $\forall x,y$ *sibling(x,y)* $\Rightarrow$ *sibling(y,x)*, which states that *sibling* is a symmetric relation. With multiple instantiation, we can also represent (limited) transitivity and recursion. As and example of limited transitivity, consider a forward reasoning system with the rule

$$\forall x,y,z\ greater\text{-}than(x,y) \wedge greater\text{-}than(y,z) \Rightarrow greater\text{-}than(x,z),$$

where *greater-than* represents the usual $>$ operator. If we have activation representing *greater-than(A,B)* and *greater-than(B,C)*, then the system will automatically conclude *greater-than(A,C)*. Transitivity and recursion are *limited* since each predicate can accommodate at most k dynamic instantiations.

4 Implementing Multiple Instantiation in the Reasoning System

4.1 Representing Predicates

Since every predicate must now be capable of representing up to k dynamic instantiations, predicates are represented using k *banks* of units. Each *bank* of an n-ary predicate P consists of τ-and nodes for the *collector* (*c:P*) and *enabler* (*e:P*) along with n ρ-btu nodes ($P_{arg_1},\ldots,P_{arg_n}$) representing the arguments of P. *Each* bank is essentially similar to the predicate representation used in [Shastri & Ajjanagadde 1990b]. Figure 2 illustrates the structure of predicates in the system. In the figure, P and Q are binary predicates while R is a three-place predicate. Note that the *enabler*, *e:P*, and the arguments, $P_{arg_1},\ldots,P_{arg_n}$, have a threshold[4] $\theta = 2$.

For a given predicate P, the *enabler* of the i-th bank $e{:}P_i$ will be active whenever the i-th bank has been instantiated with some dynamic binding. The *collector* $c{:}P_i$ of the i-th bank will be activated whenever the dynamic bindings in the i-th bank are consistent with the knowledge encoded in the system.

[4]This applies to a predicate in the backward reasoning system. In a forward reasoner, the *collector*, *c:P*, and the arguments, $P_{arg_1},\ldots,P_{arg_n}$, have a threshold $\theta = 2$.

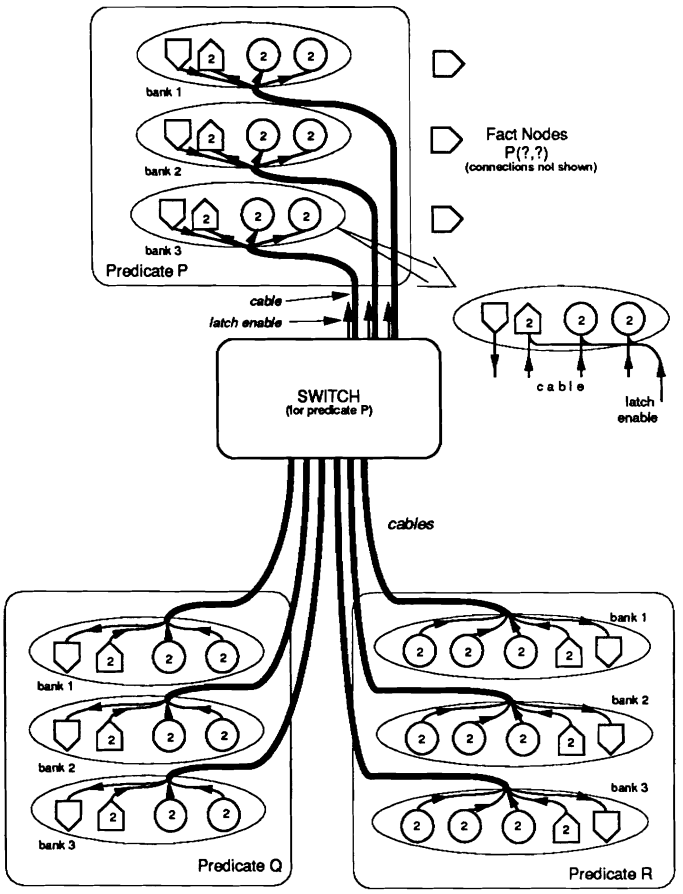


Figure 2: An overview of the multiple instantiation system. P and Q are binary predicates while R is a ternary predicate.

4.2 The Multiple Instantiation Switch

Every predicate in the extended system has an associated *multiple instantiation switch*[5], usually referred to as the *switch*. All connections to a predicate are made through its multiple instantiation switch. The switch has k output *cables* (see Figure 2), each of which connects to one bank of the predicate. A *cable* is a group of wires originating or terminating at a predicate bank; a cable, therefore, has wires from all the units (collector, enabler and argument units) in a bank. Each output cable from the switch is accompanied by a *latch enable* link[6]. Activation in the latch enable link associated with the i-th output cable indicates that the switch has successfully selected an activation for the i-th bank of the predicate, and signals the predicate bank to go ahead and represent the selected instantiation.

The interaction of the switch, the predicate banks and the inputs to the predicate are illustrated in Figure 2, which depicts the encoding of two rules: one relating P and Q and the second relating P and R. The figure assumes that the multiple instantiation constant $k = 3$ and indicates the overall connection pattern ignoring details, which will be provided later.

The switch arbitrates input instantiations to its associated predicate and brings about efficient and automatic dynamic allocation of predicate banks by ensuring the following:

[5]In this report, "switch" refers to the multiple instantiation switch, unless explicitly stated otherwise. Cf.: The switch in the type hierarchy [Mani & Shastri 1991].

[6]This *latch enable* link has nothing to do with the *latch* node mentioned in [Shastri & Ajjanagadde 1990b]. The two are entirely unrelated.

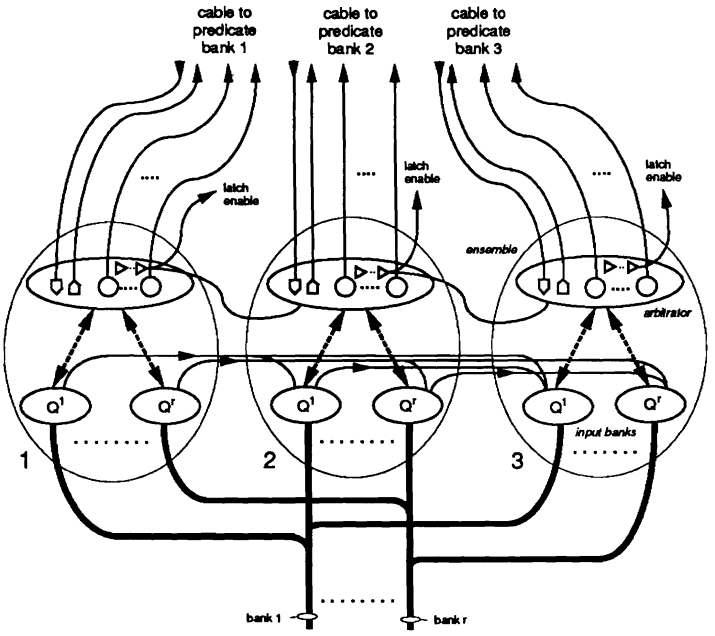


Figure 3: Structure of the multiple instantiation switch. Detailed connections are not shown to avoid cluttering.

- Fresh predicate instantiations are channeled to the predicate banks only if the predicate can accommodate more instantiations.
- All inputs that transform to the *same instantiation* are mapped into the *same predicate bank*. Thus, new instantiations selected for representation in the predicate are always unique.

4.3 Structure and Operation of the Multiple Instantiation Switch

Figures 3 and 4 illustrate the construction of the the multiple instantiation switch. The switch consists of k groups or *ensembles* of units. The figures use $k = 3$. The output of the i-th ensemble is a cable which connects to the i-th bank of the corresponding predicate. Accompanying each cable is a *latch enable* link, which signals when the instantiation selected by the switch is ready to be latched on to the predicate bank. As such, the entire switch has k cables, along with their associated latch enables, forming the output of the switch.

As can be seen in Figure 3, each ensemble consists of an *arbitrator bank*, and several *input banks*. The *arbitrator* consists of n ρ-btu nodes representing the arguments of the associated n-ary predicate, $n-1$ τ-or nodes and two τ-and nodes for the collector and enabler. Each ρ-btu node, except for the node representing the *first* argument[7], is associated with a τ-or node, as shown in Figure 4. The i-th *arbitrator bank* directly connects with the i-th bank of the predicate. The structure of the *arbitrator bank* in an ensemble, and its interaction with the *input banks*, is illustrated in Figure 4.

Figure 4 also shows the details of the *input banks*. Each *input bank* consists of n ρ-btu units representing the arguments of the predicate, and two τ-and nodes representing the collector and enabler of the bank.

[7]The enabler node *e:Arb* plays the role of the τ-or node for the first argument in the *arbitrator*. Thus, if we have a unary predicate, the *latch enable* link will originate from *e:Arb*.

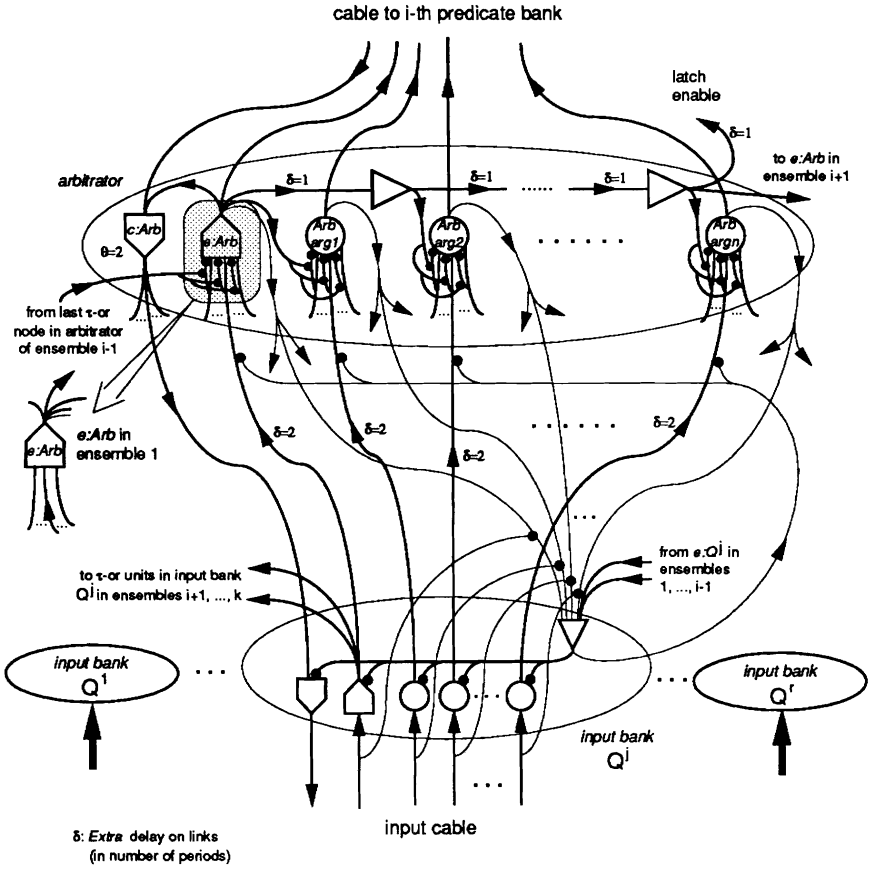


Figure 4: Structure of the i-th ensemble in the multiple instantiation switch. Only connections from *input bank* Q^j to the *arbitrator* are shown. Connections to/from other *input banks* and the *arbitrator* are implied. As indicated, connections to *e:Arb* in the first ensemble are different.

Each *input bank* also has a τ-or node associated with it. The cable terminating at the *input bank* is an input to the switch (from some bank of the consequent predicate of a rule, in which the predicate associated with the switch is in the antecedent); the output of the *input bank* connects to the *arbitrator bank* of the respective ensemble. Corresponding *input banks* across ensembles are interconnected as shown in Figures 3 and 4.

Ignoring the associated τ-or nodes, the *input banks* and the *arbitrators* have a structure which exactly mimics the bank structure of the predicate with which the switch is associated. If the predicate has n arguments, the *input banks* and *arbitrator banks* also have n ρ-btu units. The number of lines in the switch input cable is decided by the arity of the predicate originating the cable. The number of lines in the switch output depends on the arity of the predicate associated with the switch (and hence on the number of ρ-btu and τ-and nodes in the *arbitrator banks*.) Since each input cable is connected to an *input bank* in each of the k ensembles, each ensemble in the switch has the same number of *input banks*.

To start with, incoming instantiations will activate the corresponding *input banks* in all the ensembles of the switch. All ensembles in the switch *except the first* are disabled and cannot respond to incoming activation. Nodes in the *arbitrators* of all ensembles, except *e:Arb* in the first *arbitrator*, receive both an excitatory and an inhibitory input from their respective input units. The activation therefore cancels out and the *arbitrator* nodes in these other ensembles do not become active. Any activation incident on the switch will therefore affect only the first ensemble. Activation in one or more *input banks* of the first ensemble will cause the enabler in the *arbitrator*, *e:Arb*, to become active. All *input banks* with inactive enablers will be inhibited via the τ-or nodes associated with the respective *input banks*. The activation of *e:Arb* in the first

ensemble will block the inhibitory inputs to the ρ-btu node, Arb_{arg_1}, thereby enabling this node to pick a phase in which to fire in. As soon as Arb_{arg_1} selects a phase to fire in, this phase is communicated to all the *input banks*, via the τ-or node associated with the input banks (see Figure 4). For each of the *input banks*, the associate τ-or node checks if the phase selected by Arb_{arg_1} is same as the phase in which the first argument of the *input bank* is firing in. If the phases do not match, the corresponding τ-or node shuts off the entire *input bank*. Inhibitory inputs to the *arbitrator* argument nodes (from the τ-or nodes in the *input banks*) indicate which *input banks* were shut off. These direct links to the *arbitrator* facilitates the *arbitrator* to continue with its selection process without waiting for the corresponding *input banks* to turn off. Thus, when Arb_{arg_1} selects a phase ρ from its input, all activation, except that in which the first argument fires in phase ρ, is inhibited.

In the meantime, *e:Arb* would have activated the τ-or node associated with the second argument, Arb_{arg_2}, in the *arbitrator*. This causes Arb_{arg_2} to select a phase from the activation remaining after inhibiting instantiation that do not agree with Arb_{arg_1}. Note that Arb_{arg_2} is enabled by the associated τ-or node *independent* of Arb_{arg_1}. Thus, Arb_{arg_2} will select a phase to fire in *even if* Arb_{arg_1} *is inactive*. Arb_{arg_1} would remain inactive if all incoming instantiations have an unbound first argument.

The process continues, allowing Arb_{arg_3}, ..., Arb_{arg_n} to select phases in which to fire in. After, Arb_{arg_n} has made its choice, the first ensemble would have picked an instantiation to be channeled to the predicate bank. The *latch enable*, which originates at the τ-or node associated with Arb_{arg_n}, becomes active and the selected instantiation is transferred to the first predicate bank. Also, a link from this last τ-or node to *e:Arb* in the second ensemble enables the second ensemble to select a fresh instantiation.

After the first ensemble has selected an instantiation to be channeled to the predicate, only those *input banks* which represent this exact pattern of activation will be active in the first ensemble. All other *input banks* will be inhibited due to some mismatch in the firing pattern. Further, *input banks* remaining active in the first ensemble will blot out activation in all corresponding *input banks* in all the other $(2, \ldots, k)$ ensembles. This ensures that the instantiation selected by the first ensemble will not be selected again in any other ensemble.

Once the second ensemble is enabled (by blocking inhibitory inputs to *e:Arb* in the ensemble), it will pick an instantiation, channel it to the predicate bank, and enable the third ensemble in the switch. Again, only *input banks* with an activation pattern identical to the one chosen by the arbitrator of the second ensemble will remain active in this ensemble, and these will inhibit corresponding *input banks* in the other $(3, \ldots, k)$ ensembles. The third ensemble will then pick an instantiation, and so on. The process continues until k instantiations have been channeled to the predicate, after which, any fresh input instantiations are ignored.

Note that the ensembles in the switch have an implicit ordering from left to right (Figure 3). The i-th ensemble has priority over the $i+1, \ldots, k$ ensembles, in that the i-th ensemble gets to pick an instantiation before any of the $i+1, \ldots, k$ ensembles get to pick theirs.

At any point in time, if the i-th ensemble ($1 \leq i \leq k$) is making its choice, it will always select an instantiation which is *different* from those picked by the first $i-1$ instantiations. Further, a new instantiation arriving at the switch will be checked to see if it has already been assigned a bank in the predicate. If so, the activation will be diverted to the bank already assigned to it. If not, the activation is assigned a new bank in the predicate, via the next unused ensemble in the switch. Thus, all the instantiations channeled to the predicate are unique.

Further, whenever the collector of the i-th bank of the predicate associated with the switch becomes active, the activation gets transmitted to *c:Arb* in switch ensemble i. Activation of *c:Arb* is distributed to the active *input banks* in the ensemble, and turns on the collector of the predicate bank which *originated the instantiation* selected by the i-th ensemble. The activation of *c:Arb* received by inactive *input banks* in the ensemble is killed by the active τ-or nodes associated with the corresponding *input banks*. Also note that the link from *e:Arb* to *c:Arb* ensures that *c:Arb* becomes active only if *both e:Arb and* the collector of the associated predicate bank are simultaneously active.

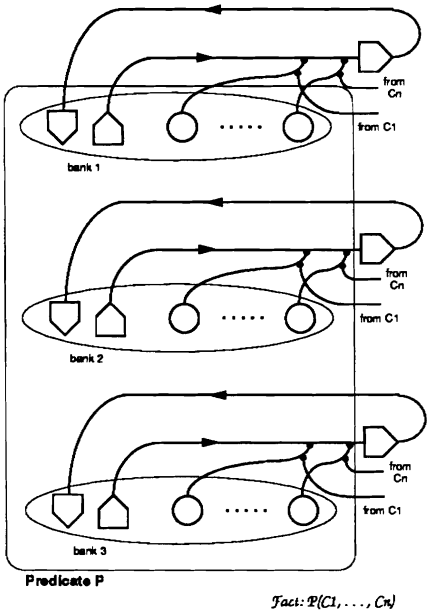


Figure 5: Encoding long term facts in the multiple instantiation reasoner. Fact encoded: $P(C_1, \ldots, C_n)$.

5 Encoding Rules and Facts

Every predicate in the system has k banks of units for representing k instantiations. Further every predicate has an associated multiple instantiation switch which arbitrates the instantiations which will be represented in the predicate. Given this modified underlying framework, we shall now look at how rules and long-term facts can be encoded in this extended system. All we need to do is extend the scheme of [Shastri & Ajjanagadde 1990b] to accommodate the modifications mentioned earlier.

5.1 Encoding Facts

For a given predicate, since the dynamic instantiation which matches a long-term fact could occur in *any one* of the k banks for that predicate, we need a fact-pattern-matcher for *each* of the predicate banks. Thus, any fact of the form $P(C_1, \ldots, C_n)$ will be encoded using k τ-and nodes — one for each bank of P — as illustrated in Figure 5[8]. This is an obvious extension of the fact encoding scheme of [Shastri & Ajjanagadde 1990b] to handle multiple instantiation.

5.2 Encoding Rules

Figure 2 illustrates rule encoding at a very gross level. Figure 6 is a more detailed description of the way rules are encoded in the extended system. Figure 6 depicts the encoding of the rule $\forall x, y$ *P(x,y)* $\Rightarrow$ *Q(y,x)*.

Each bank of predicate Q is connected to an *input bank* in every ensemble of the switch for P. Thus, the k banks of predicate Q require a total of k^2 *input banks* — k *input banks* in each of the k ensembles of the switch.

Consider the connection from the i-th bank of Q to the corresponding *input bank* in the j-th ensemble of the switch for P. As mentioned earlier, the *input bank* has a structure identical to the bank structure of predicate P. The input cable from bank i of Q connects to the *input bank* as though the *input bank*

[8]If the system also includes the type hierarchy [Mani & Shastri 1991], where constants are encoded as clusters of nodes, the inhibitory links from $C_1, \ldots, C_n$ in Figure 5 would be bundles of k wires instead of single links.

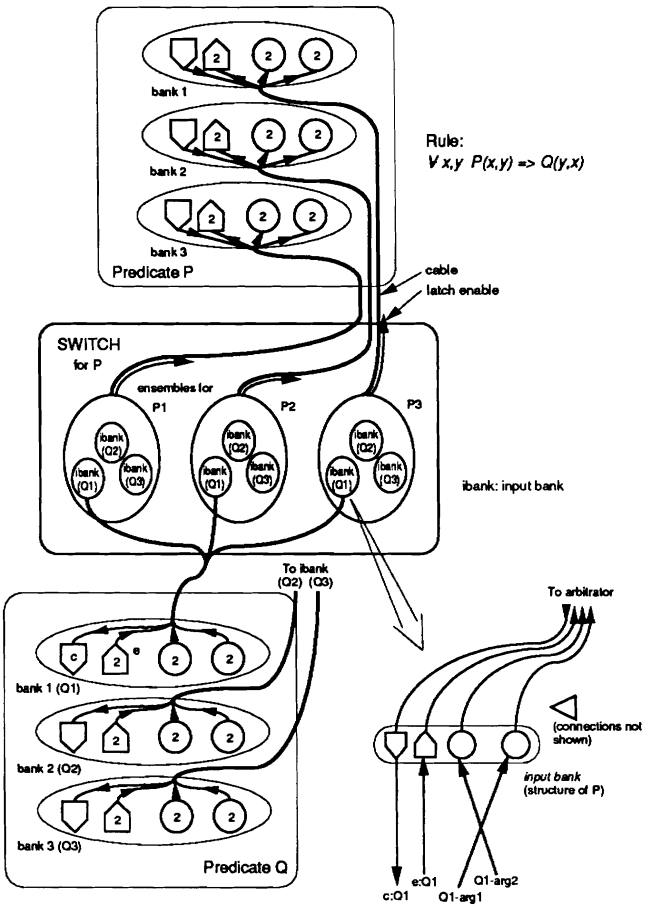


Figure 6: Encoding rules in the multiple instantiation reasoner. Rule encoded: $\forall x, y\ P(x,y) \Rightarrow Q(y,x)$. To avoid cluttering, only part of the connections are indicated.

itself represented the predicate P. Thus, the connection pattern between the bank of Q and the *input bank* of the switch for P is identical to the connection pattern between the actual predicates in the system of [Shastri & Ajjanagadde 1990b]. In particular, we have the following connections for all $1 \leq i \leq k$ and $1 \leq j \leq k$:

- The enabler $e{:}Q_i$ of the i-th bank of Q is connected to the *enabler* in the corresponding *input bank* of the j-th ensemble of the switch for P.
- The *collector* in the same *input bank* is linked to $c{:}Q_i$, the collector of the i-th bank of Q.
- The first argument of the i-th bank of Q connects to the second argument in the *input bank* while the second argument of the predicate bank connects to the first argument of the *input bank* (see Figure 6).

Now, since each bank of Q connects to an *input bank* in every ensemble of the switch for P, it follows that the i-th bank of Q is connected to an *input bank* in every ensemble of the switch (Figure 6). As a consequence, the collector $c{:}Q_i$ of the i-th bank of Q receives input from the respective collectors in the *input banks* of *all* the ensembles of the switch. The τ-or unit associated with the *input bank* ensures that the collector turns on if and only if the instantiation received by the *input bank* has been channeled to the predicate, *and* the collector in the corresponding bank of predicate P is active (refer to Figure 4). In other words, the predicate collector $c{:}Q_i$ would be activated if:

- The activation of Q_i, the i-th bank of Q, has been channeled to P_j, the j-th bank of P; and

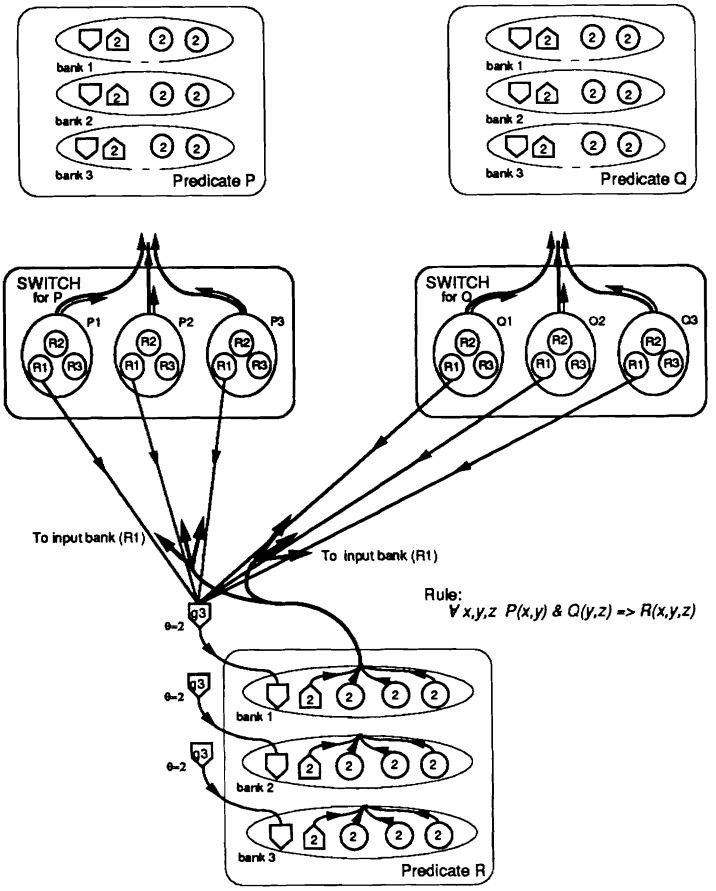


Figure 7: Encoding rules with multiple predicates in the antecedent. Rule encoded: $\forall x, y, z\ P(x,y) \wedge Q(y,z) \Rightarrow R(x,y,z)$. To avoid cluttering, only relevant connections are indicated.

- The collector $c{:}P_j$ is active.

The combination of collectors in the *arbitrator* and the *input bank* therefore serve as a mechanism for transmitting the state of the collector $c{:}P_j$ to the collector $c{:}Q_i$ of predicate Q.

Rules with multiple predicates in the antecedent are handled by an obvious extension of the above procedure. Figure 7 gives a network which encodes the rule $\forall x, y, z\ P(x,y) \wedge Q(y,z) \Rightarrow R(x,y,z)$. The $g3_i$[9] nodes check that the dynamic activation of the i-th bank of R is consistent with facts for *both* P and Q. Note that $g3_i$ will become active irrespective of which banks of P and Q contain the activation which triggered the required facts.

The encoding of a rule with repeated variables in the consequent, existential variables in the consequent and constants in the consequent is shown in Figure 8. The output of the $g1$ nodes inhibit the links from Q to the predicate(s) in the antecedent of the rule. Note that the scheme is identical to the one used to handle such conditions in [Shastri & Ajjanagadde 1990b], except that we repeat the scheme for each of the k banks[10].

[9] $g3_i$ refers to the $g3$ node for the i-th bank of the predicate.

[10] Here again, the inhibitory links from constants would be bundles of k links if we are using the type hierarchy of [Mani & Shastri 1991].

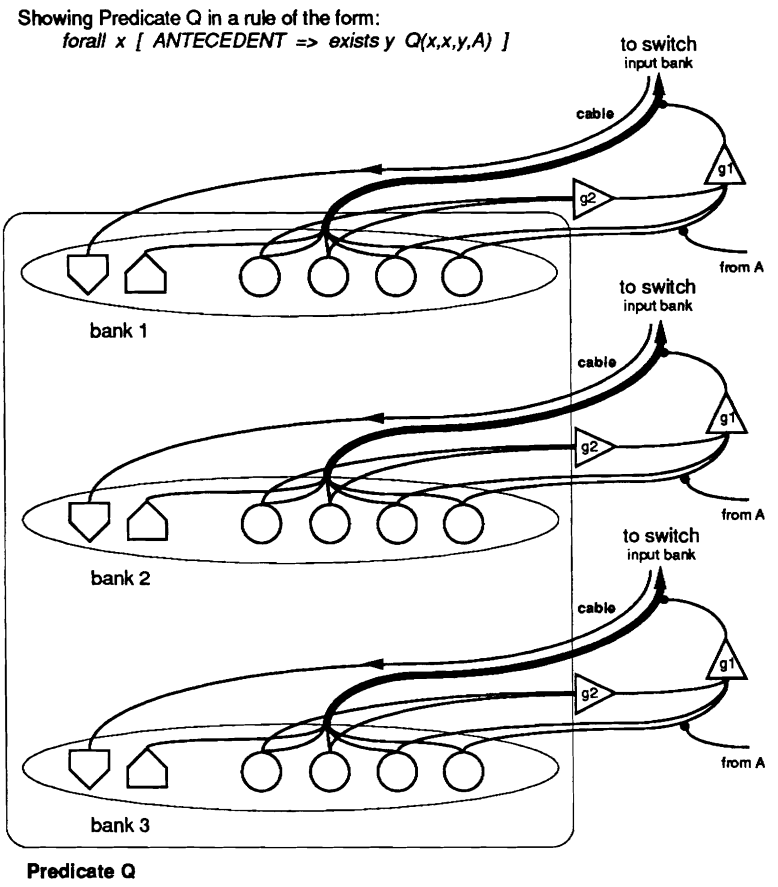


Figure 8: Encoding rules with special conditions (repeated variables, existentially quantified variables and constants) in the consequent. Rule encoded: $\forall x$ ANTECEDENT $\Rightarrow \exists y$ *Q(x,x,y,A)*.

Complexity of the Network

The extended reasoning system requires $O(\mathcal{C} + \mathcal{F} + \mathcal{P})$ nodes and links, where $\mathcal{C}$ is the total number of constants used in the system, $\mathcal{F}$ is the total number of long-term facts present in the reasoner, and $\mathcal{P}$ is the sum of the arities of all predicates in the rule base. The constant of proportionality for the network complexity is proportional to k^2, where k is the multiple instantiation constant. Thus, as in [Shastri & Ajjanagadde 1990b], the network complexity is *at most linear* in the size of the knowledge base although the constant of proportionality is now larger.

A similar comment holds for the time complexity: the system can still answer queries in time proportional to the length of the shortest derivation [Shastri & Ajjanagadde 1990b]; but now, the constant of proportionality is slightly larger, since we also need to consider the time required for the activation to propagate through the switches. Given a predicate P, the best case propagation time for activation passing through its switch is proportional to n, the arity of P; in the worst case, propagation time is proportional to kn. If we assume that n_{max} is the *maximum* arity of any predicate in the reasoning system, then the constant of proportionality for the time complexity will be proportional to n_{max} (in the best case) or kn_{max} (in the worst case), *irrespective* of the predicate under consideration.

6 Multiple Instantiation in a Forward Reasoning System

All along, we have looked at how the basic system proposed in [Shastri & Ajjanagadde 1990b] could be extended to accommodate multiple instantiation of predicates in the *backward reasoner*. We now consider issues that arise when incorporating multiple instantiation of predicates in the *forward reasoner*.

In a forward reasoning system, predicates have the same structure as for the backward reasoning system. As before, every predicate has an associated multiple instantiation switch[11]. Rules with a single predicate in the antecedent can be encoded directly: ***each*** bank of the antecedent predicate is connected to ***input banks*** in every ensemble of the switch for the consequent predicate. Rules with multiple predicates in the antecedent, however, require special consideration. Suppose we have a rule of the form: $\forall x, y, z$ *P(x,y)* $\wedge$ *Q(y,z)* $\Rightarrow$ *R(x,y,z)*. Suppose also that we are given the dynamic facts *P(A,B)* and *Q(B,C)*. Then we should be able to conclude *R(A,B,C)*. But the dynamic fact *P(A,B)* could be represented in *any* of the k banks allocated for P. Similarly *Q(B,C)* could be active in *any* of the k banks allocated for Q. Hence to conclude *R(A,B,C)*, we would need to pair each bank of P with all the banks of Q and check if the second argument of P is the same as the first argument of Q; in other words, we need to check if the second argument of P_i is the same as the first argument of Q_j for $1 \leq i, j \leq k$. The obvious solution to this problem requires $O(k^m)$ nodes and links to encode each multiple antecedent rule, where m is the number of predicates in the antecedent of the rule and k is the multiple instantiation constant. Typically, we expect the value of k to be around 3 (as argued in [Shastri & Ajjanagadde 1990b]), and m to be around 2. Generally, in a rule containing an antecedent with several predicates, most of the predicates function to specify constraints on the arguments of one or two key predicates. Since the reasoning system (combined with the type hierarchy introduced in [Mani & Shastri 1991]) can handle rules with typed variables, most of the predicates enforcing type constraints can be replaced by typed variables. For example, the rule

$\forall x, y$ *collide(x,y)* $\wedge$ *animate(x)* $\wedge$ *solidobj(y)* $\Rightarrow$ *hurt(x)*

with three predicates in the antecedent is equivalent to the simple rule:

$\forall$ *x:animate, y:solidobj collide(x,y)* $\Rightarrow$ *hurt(x).*

The latter rule can be directly encoded in the extended reasoning system. Even if this "compression" of the antecedent were not possible, we could always introduce dummy predicates and split a rule with several predicates in the antecedent into several rules with just a few predicates in the antecedent. Thus, with typical values of $k \approx 3$ and $m \approx 2$, the extra cost of encoding rules in the forward reasoner with multiple instantiation is a factor of about 10 ($\approx 3^2$).

Special conditions in a rule (like repeated variables, existential variables in the antecedent, constants in the antecedent, etc.) can be handled as usual [Shastri & Ajjanagadde 1990b], *before* connecting a predicate bank to the *input banks* in the switch.

Incorporating multiple instantiation into the forward reasoner gives us the capability to encode rules like:

$\forall x, y, z$ *loves(x,y)* $\wedge$ *loves(y,z)* $\Rightarrow$ *jealous(x,z),*

and infer *jealous(John,Tom)* given *loves(John,Mary)* and *loves(Mary,Tom)*.

[11] Though the multiple instantiation switch associated with a predicate in the forward reasoning system is structurally identical to the switch used in the backward reasoner, there are a few minor functional differences. Figures 3 and 4 show connections in the context of a backward reasoning system. In a forward reasoner, the enabler in the *arbitrator*, *e:Arb*, connects to the *collector* of the associated predicate, while the enablers in the *input banks* receive inputs from the *collectors* of the input predicate banks. The collectors in the *arbitrator* and *input banks* are left unconnected, as are the enablers in the predicate banks.

7 Simulations

The appendices include screen dumps from simulations of the multiple instantiation reasoning system, combined with the type hierarchy. This assembly of figures provides a trace of the time course of propagating activation in the system. The figures also provide simulation snap-shots of the functioning of the multiple instantiation and type hierarchy switches.

8 Conclusions

Extending the connectionist rule-based reasoning system to accommodate multiple instantiation of predicates enables the system to handle a wider and more powerful set of rules, facts and queries. The extended system can encode and reason with rules that capture symmetry, transitivity and recursion, provided the number of multiple instantiations required to draw a conclusion remains bounded.

The multiple instantiation reasoning system has been combined with a connectionist type hierarchy [Mani & Shastri 1991] to provide a more flexible and powerful system. All these features have been successfully included in the simulation system reported in [Mani 1990].

9 Acknowledgements

This work has benefited from discussions and presentations at the Connectionist Group meetings at Penn. Special thanks to Jeff Aaronson for discussions on an earlier version of the multiple instantiation switch.

A Simulations: The Backward Reasoning System

The figures in the following pages show the progress of activation in the simulation of a backward reasoning system interfaced with a type hierarchy. The rule-base encodes the following rule and fact:

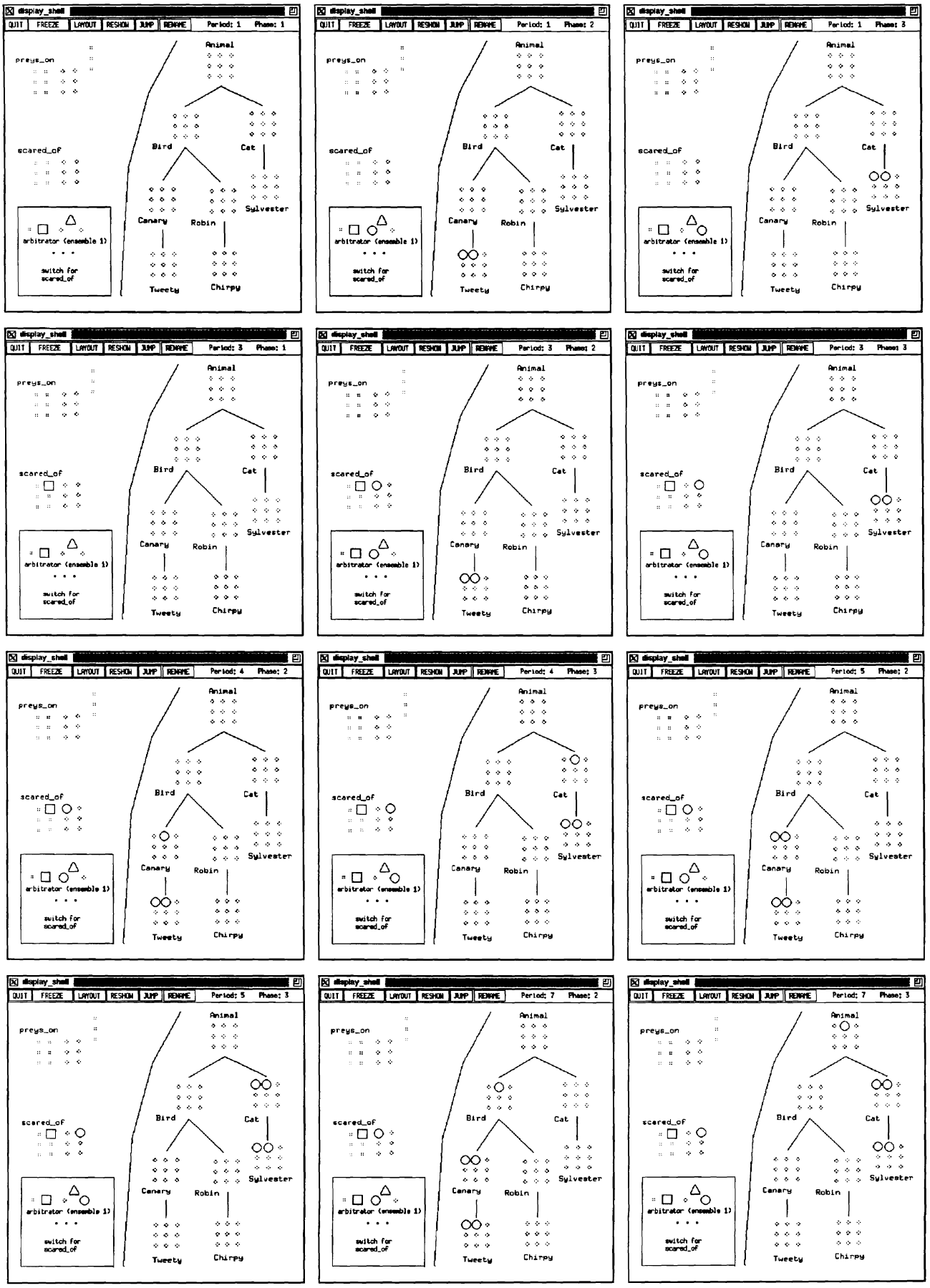
$\forall x, y$ *preys-on(x,y)* $\Rightarrow$ *scared-of(y,x)*
$\forall$*x:Cat, y:Bird preys-on(x,y).*

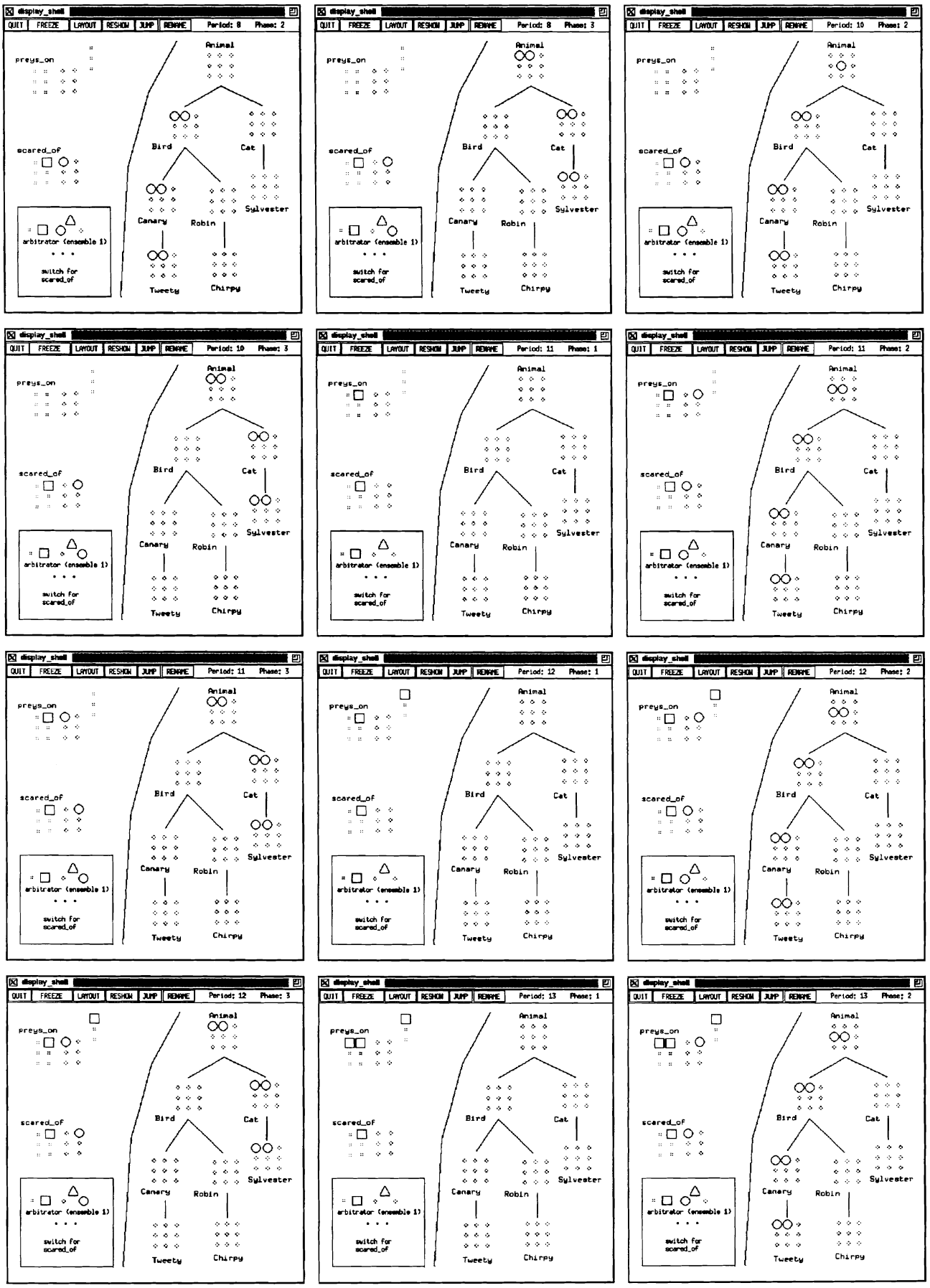
The type hierarchy encodes the following information:

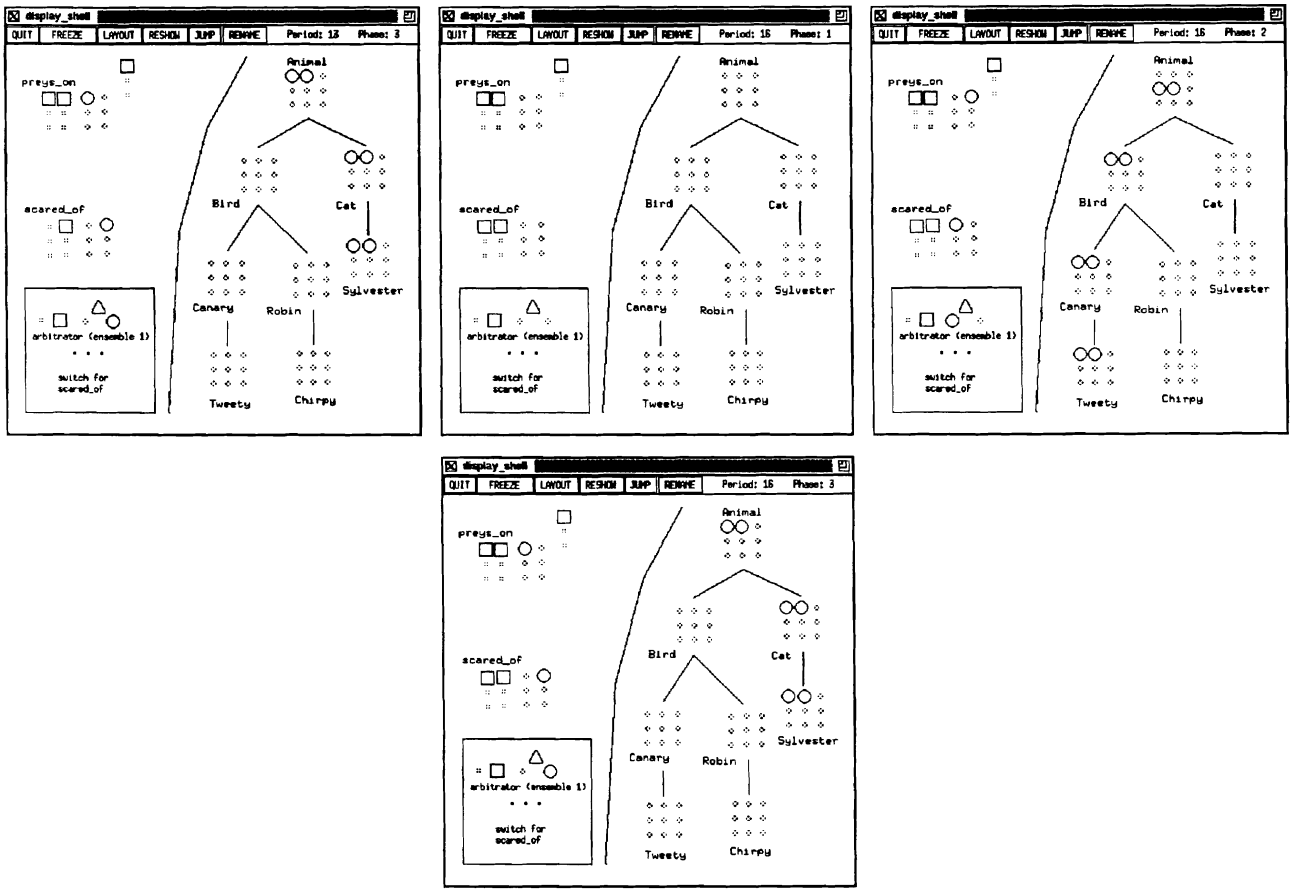
is-a(Bird,Animal)
is-a(Cat,Animal)
is-a(Robin,Bird)
is-a(Canary,Bird)
is-a(Chirpy,Robin)
is-a(Tweety,Canary)
is-a(Sylvester,Cat).

The query posed is: *scared-of(Tweety,Sylvester)?* ("Is Tweety scared of Sylvester?"). Note that the query is posed by activating nodes in the *arbitrator* bank of the switch associated with the *scared-of* predicate.

The figures are arranged such that time increases left to right and top to bottom. Each figure is a snapshot of the simulation at a particular time instant. Each time instant is uniquely identified by a *period* and *phase* in the simulation. This information is displayed at the top right part of each display. Only relevant periods and phases are shown. Activity during periods when activation is propagating through switches (the multiple instantiation switch and/or the type hierarchy switch) is not shown. Appendices B and C document this activity. τ-and nodes are represented as squares, ρ-btu nodes as circles and τ-or nodes as triangles. The left half of each figure displays relevant units in the rule-base while the right half contains units forming the concepts in the type hierarchy.


display_shell
QUIT
FREEZE
LAYOUT
RESHOW
JUMP
RENAME
Period: 1 Phase: 1
Period: 1 Phase: 2
Period: 1 Phase: 3
Period: 3 Phase: 1
Period: 3 Phase: 2
Period: 3 Phase: 3
Period: 4 Phase: 2
Period: 4 Phase: 3
Period: 5 Phase: 2
Period: 5 Phase: 3
Period: 7 Phase: 2
Period: 7 Phase: 3
preys_on
scared_of
arbitrator (ensemble 1)
switch for scared_of
Animal
Bird
Cat
Canary
Robin
Sylvester
Tweety
Chirpy

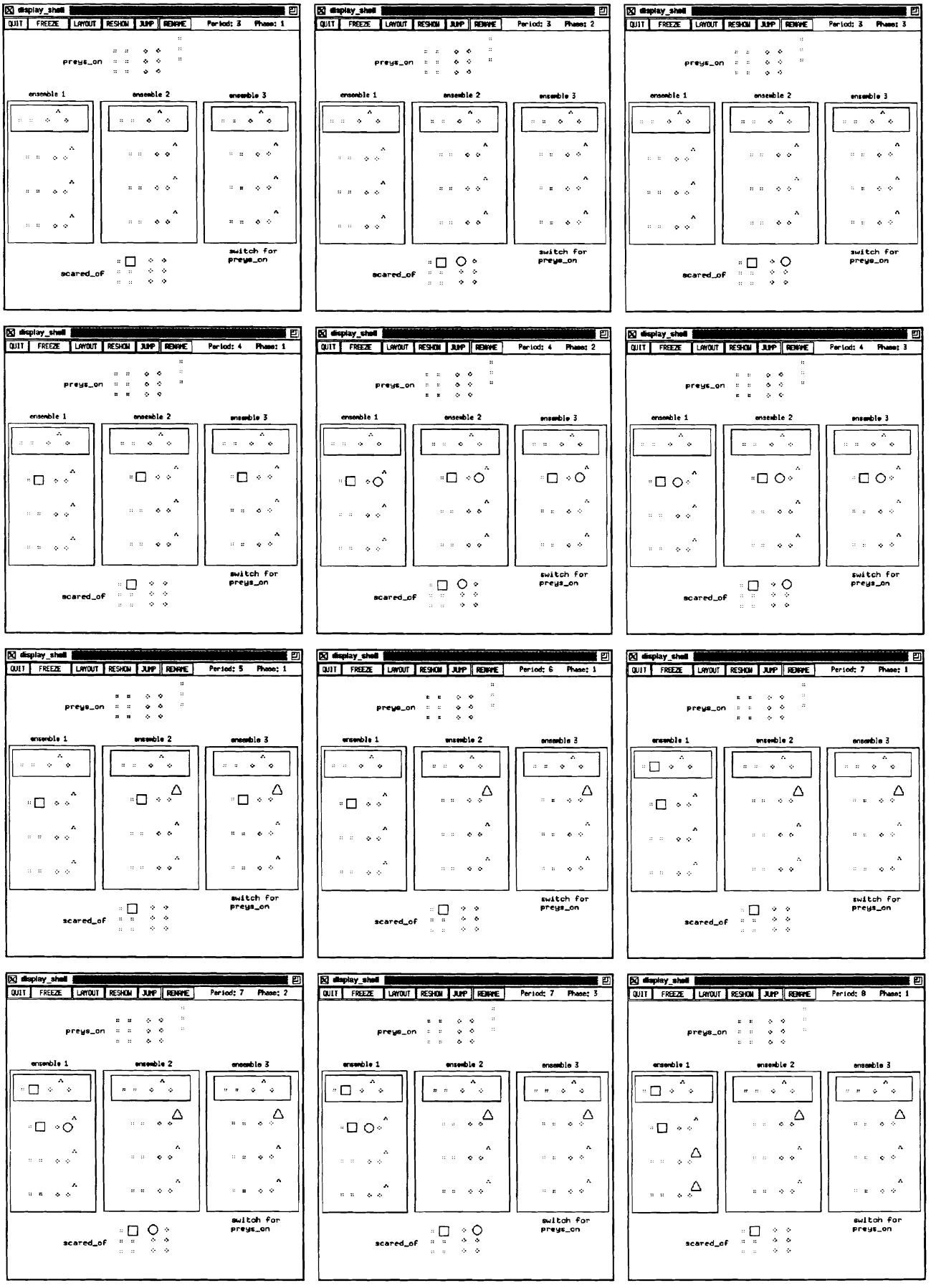

display_shell
QUIT
FREEZE
LAYOUT
RESHOW
JUMP
RENAME
Period: 8 Phase: 2
Period: 8 Phase: 3
Period: 10 Phase: 2
Period: 10 Phase: 3
Period: 11 Phase: 1
Period: 11 Phase: 2
Period: 11 Phase: 3
Period: 12 Phase: 1
Period: 12 Phase: 2
Period: 12 Phase: 3
Period: 13 Phase: 1
Period: 13 Phase: 2
preys_on
scared_of
Animal
Bird
Cat
Canary
Robin
Sylvester
Tweety
Chirpy
arbitrator (ensemble 1)
switch for scared_of

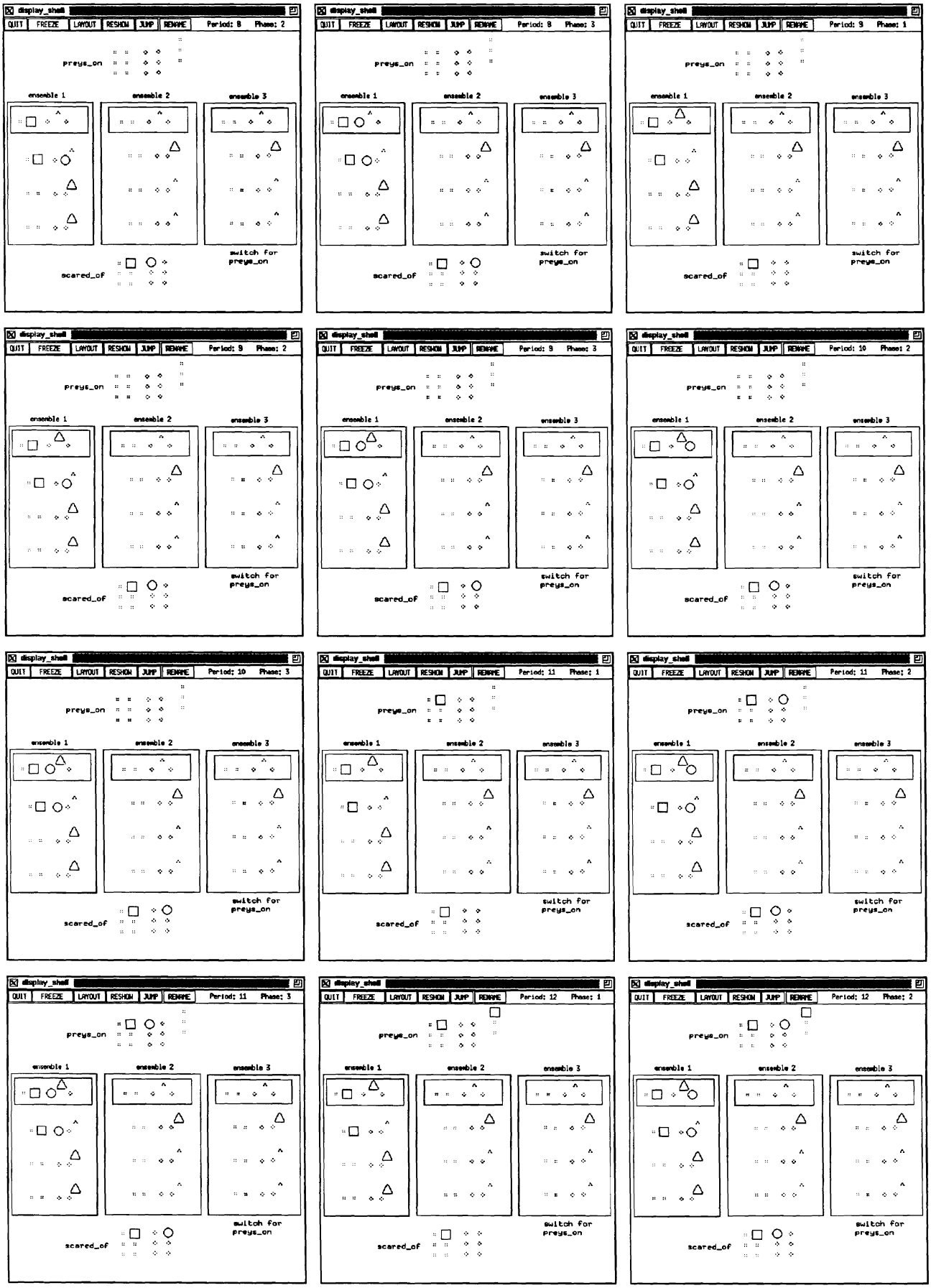

display_shell
QUIT
FREEZE
LAYOUT
RESHOW
JUMP
RENAME
Period: 13
Phase: 3
preys_on
scared_of
arbitrator (ensemble 1)
switch for scared_of
Animal
Bird
Cat
Canary
Robin
Sylvester
Tweety
Chirpy
display_shell
QUIT
FREEZE
LAYOUT
RESHOW
JUMP
RENAME
Period: 16
Phase: 1
preys_on
scared_of
arbitrator (ensemble 1)
switch for scared_of
Animal
Bird
Cat
Canary
Robin
Sylvester
Tweety
Chirpy
display_shell
QUIT
FREEZE
LAYOUT
RESHOW
JUMP
RENAME
Period: 16
Phase: 2
preys_on
scared_of
arbitrator (ensemble 1)
switch for scared_of
Animal
Bird
Cat
Canary
Robin
Sylvester
Tweety
Chirpy
display_shell
QUIT
FREEZE
LAYOUT
RESHOW
JUMP
RENAME
Period: 16
Phase: 3
preys_on
scared_of
arbitrator (ensemble 1)
switch for scared_of
Animal
Bird
Cat
Canary
Robin
Sylvester
Tweety
Chirpy

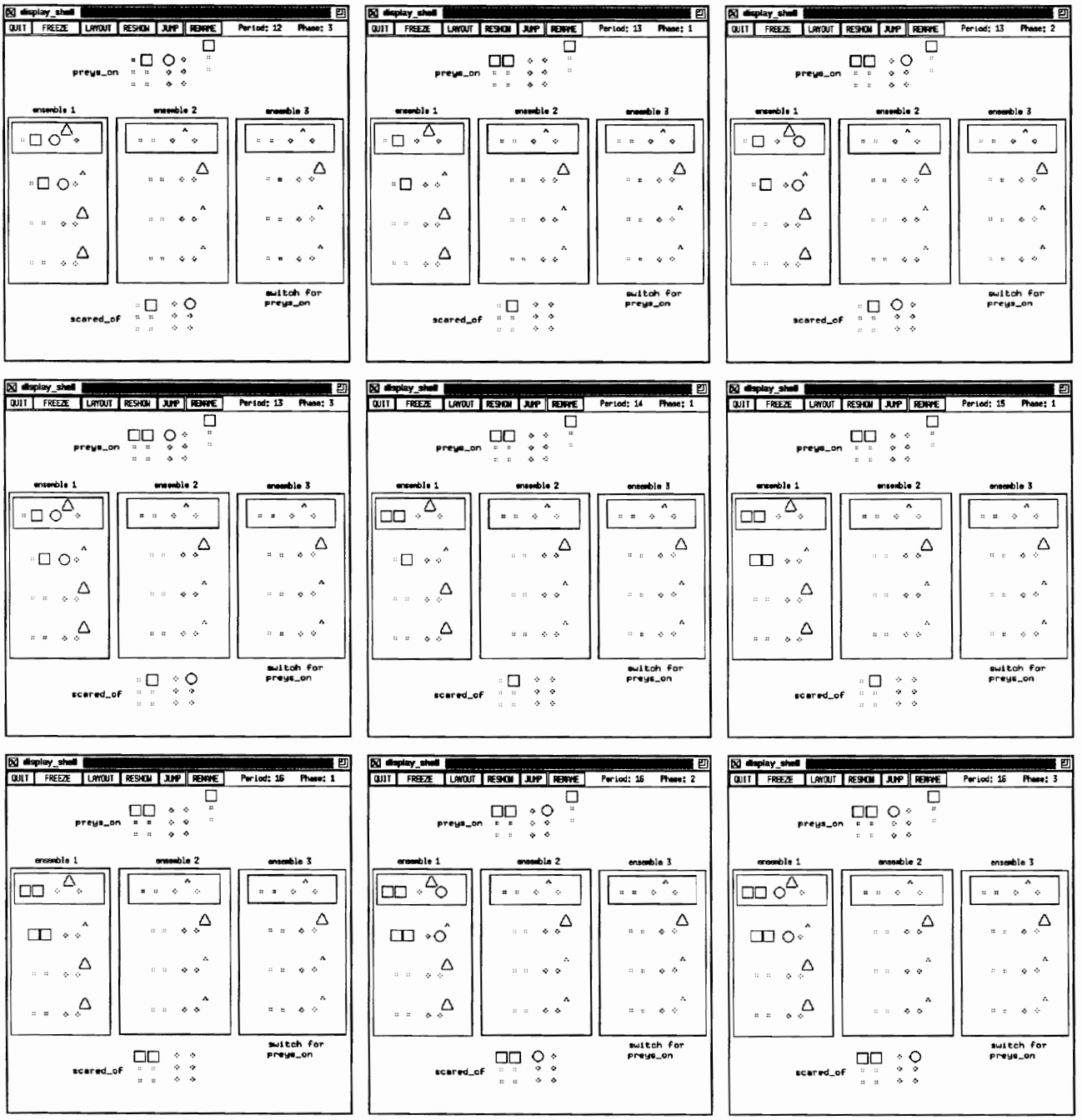
B Simulations: The Multiple Instantiation Switch

Figures in this appendix demonstrate the propagation of activation in the multiple instantiation switch. The rules and facts encoded by the network are listed in Appendix A. The switch shown is associated with the *preys-on* predicate in the rule base. Activation originating in the *scared-of* predicate spreads to a bank of the *preys-on* predicate via this switch. This activation is a result of the query *scared-of(Tweety,Sylvester)?* (see Appendix A).

Figures are arranged such that time increases left to right and top to bottom. Each figure is a snap-shot of the simulation at a particular time instant. Each time instant is uniquely identified by a *period* and *phase* in the simulation. This information is displayed at the top right part of each display. Only relevant periods and phases are shown. τ-and nodes are represented as squares, ρ-btu nodes as circles and τ-or nodes as triangles. The ***arbitrator*** bank of each ensemble in the switch is enclosed in a bounding box. *Input banks* are shown below the ***arbitrator***. Due to space limitations on the display screen, *input banks* in each ensemble are lined up vertically (unlike in Figures 3 and 4, where they are lined up horizontally).


display_shell
QUIT
FREEZE
LAYOUT
RESHOW
JUMP
RENAME
Period: 3 Phase: 1
Period: 3 Phase: 2
Period: 3 Phase: 3
Period: 4 Phase: 1
Period: 4 Phase: 2
Period: 4 Phase: 3
Period: 5 Phase: 1
Period: 6 Phase: 1
Period: 7 Phase: 1
Period: 7 Phase: 2
Period: 7 Phase: 3
Period: 8 Phase: 1
preys_on
ensemble 1
ensemble 2
ensemble 3
scared_of
switch for preys_on

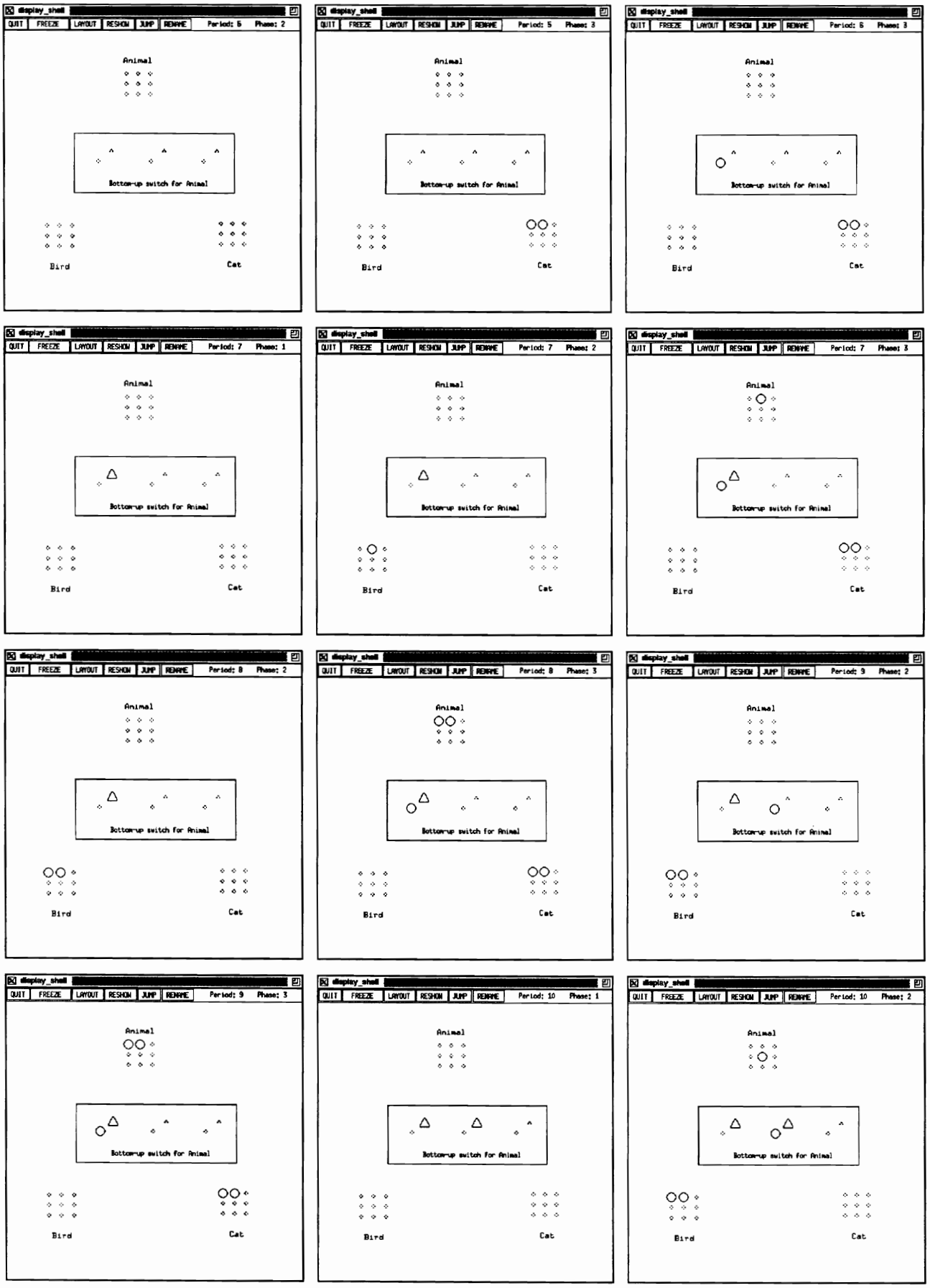

display_shell
QUIT
FREEZE
LAYOUT
RESHOW
JUMP
RENAME
Period: 8 Phase: 2
Period: 8 Phase: 3
Period: 9 Phase: 1
Period: 9 Phase: 2
Period: 9 Phase: 3
Period: 10 Phase: 2
Period: 10 Phase: 3
Period: 11 Phase: 1
Period: 11 Phase: 2
Period: 11 Phase: 3
Period: 12 Phase: 1
Period: 12 Phase: 2
preys_on
ensemble 1
ensemble 2
ensemble 3
scared_of
switch for preys_on

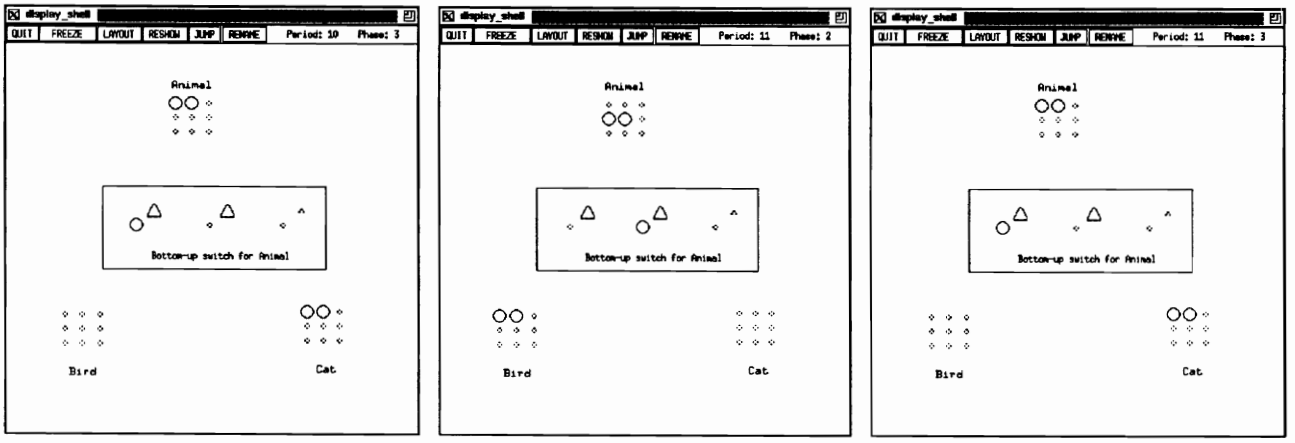

display_shell
QUIT
FREEZE
LAYOUT
RESHOW
JUMP
RENAME
Period: 12 Phase: 3
Period: 13 Phase: 1
Period: 13 Phase: 2
Period: 13 Phase: 3
Period: 14 Phase: 1
Period: 15 Phase: 1
Period: 16 Phase: 1
Period: 16 Phase: 2
Period: 16 Phase: 3
preys_on
ensemble 1
ensemble 2
ensemble 3
switch for preys_on
scared_of

C Simulations: The Type Hierarchy Switch

Figures in this appendix demonstrate the propagation of activation in the type hierarchy switch. The facts encoded in the type hierarchy is listed in Appendix A. The switch shown is associated with the *Animal* concept. The simulation indicates how the switch handles bottom-up activation coming from both the *Cat* and *Bird* concepts and assigns them to banks in *Animal*. The structure and operation of the switch are described in [Mani & Shastri 1991].

Figures are arranged such that time increases left to right and top to bottom. Each figure is a snap-shot of the simulation at a particular time instant. Each time instant is uniquely identified by a *period* and *phase* in the simulation. This information is displayed at the top right part of each display. Only relevant periods and phases are shown. ρ-btu nodes are represented as circles and τ-or nodes as triangles.


display_shell
QUIT FREEZE LAYOUT RESHOW JUMP RENAME Period: 5 Phase: 2
Animal
Bottom-up switch for Animal
Bird
Cat
display_shell
QUIT FREEZE LAYOUT RESHOW JUMP RENAME Period: 5 Phase: 3
Animal
Bottom-up switch for Animal
Bird
Cat
display_shell
QUIT FREEZE LAYOUT RESHOW JUMP RENAME Period: 6 Phase: 3
Animal
Bottom-up switch for Animal
Bird
Cat
display_shell
QUIT FREEZE LAYOUT RESHOW JUMP RENAME Period: 7 Phase: 1
Animal
Bottom-up switch for Animal
Bird
Cat
display_shell
QUIT FREEZE LAYOUT RESHOW JUMP RENAME Period: 7 Phase: 2
Animal
Bottom-up switch for Animal
Bird
Cat
display_shell
QUIT FREEZE LAYOUT RESHOW JUMP RENAME Period: 7 Phase: 3
Animal
Bottom-up switch for Animal
Bird
Cat
display_shell
QUIT FREEZE LAYOUT RESHOW JUMP RENAME Period: 8 Phase: 2
Animal
Bottom-up switch for Animal
Bird
Cat
display_shell
QUIT FREEZE LAYOUT RESHOW JUMP RENAME Period: 8 Phase: 3
Animal
Bottom-up switch for Animal
Bird
Cat
display_shell
QUIT FREEZE LAYOUT RESHOW JUMP RENAME Period: 9 Phase: 2
Animal
Bottom-up switch for Animal
Bird
Cat
display_shell
QUIT FREEZE LAYOUT RESHOW JUMP RENAME Period: 9 Phase: 3
Animal
Bottom-up switch for Animal
Bird
Cat
display_shell
QUIT FREEZE LAYOUT RESHOW JUMP RENAME Period: 10 Phase: 1
Animal
Bottom-up switch for Animal
Bird
Cat
display_shell
QUIT FREEZE LAYOUT RESHOW JUMP RENAME Period: 10 Phase: 2
Animal
Bottom-up switch for Animal
Bird
Cat


display_shell
QUIT
FREEZE
LAYOUT
RESHOW
JUMP
RENAME
Period: 10
Phase: 3
Animal
Bottom-up switch for Animal
Bird
Cat
display_shell
QUIT
FREEZE
LAYOUT
RESHOW
JUMP
RENAME
Period: 11
Phase: 2
Animal
Bottom-up switch for Animal
Bird
Cat
display_shell
QUIT
FREEZE
LAYOUT
RESHOW
JUMP
RENAME
Period: 11
Phase: 3
Animal
Bottom-up switch for Animal
Bird
Cat

References

[Ajjanagadde & Shastri 1991] Ajjanagadde, V., and Shastri, L. 1991. Rules and variables in neural nets. *Neural Computation*, **3**:121–134.

[Feldman 1982] Feldman, J. A. 1982. Dynamic connections in neural networks. *Bio-Cybernetics*, **46**:27–39.

[Malsburg 1986] von der Malsburg, C. 1986. Am I thinking assemblies? In G. Palm and A. Aertsen (Eds.). *Brain Theory*. Berlin: Springer-Verlag.

[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 Connectionist Type Hierarchy with a Connectionist Rule-Based Reasoner. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, 418–423. Hillsdale NJ: Lawrence Erlbaum Associates.

[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 MA: 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.