January 1989

A Connectionist System for Rule Based Reasoning With Multi-Place Predicates and Variables

Lokendra Shastri
University of Pennsylvania

Venkat Ajjanagadde
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-89-06.

A Connectionist System for Rule Based Reasoning With Multi-Place Predicates and Variables

Abstract

McCarthy has observed that the representational power of most connectionist systems is restricted to unary predicates applied to a fixed object. More recently, Fodor and Pylyshyn have made a sweeping claim that connectionist systems cannot incorporate systematicity and compositionality. These comments suggest that representing structured knowledge in a connectionist network and using this knowledge in a systematic way is considered difficult if not impossible. The work reported in this paper demonstrates that a connectionist system can not only represent structured knowledge and display systematic behavior, but it can also do so with extreme efficiency. The paper describes a connectionist system that can represent knowledge expressed as *rules* and *facts* involving *multi-place* predicates (i.e., *n-ary* relations), and draw limited, but sound, inferences based on this knowledge. The system is extremely efficient - in fact, optimal, as it draws conclusions in time proportional to the *length* of the proof. It is observed that representing and reasoning with structured knowledge requires a solution to the *variable binding* problem. A solution to this problem using a *multi-phase* clock is proposed. The solution allows the system to maintain and propagate an arbitrary number of variable bindings during the reasoning process. The work also identifies constraints on the structure of inferential dependencies and the nature of quantification in individual rules that are required for efficient reasoning. These constraints may eventually help in modelling the remarkable human ability of performing certain inferences with extreme efficiency.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-89-06.

A CONNECTIONIST SYSTEM FOR RULE BASED REASONING WITH MULTI-PLACE PREDICATES AND VARIABLES

Lokendra Shastri
Venkat Ajjanagadde

MS-CIS-89-06
LINC LAB 141

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

January 1989

Running head: A Connectionist System for Rule-Based Reasoning

---

Acknowledgements: This research was supported in part by DARPA grant NOOO14-85-K-0018, NOOO14-85-K-0807, NSF grants MCS-8219196-CER, IRI84-10413-AO2, IRI 88-05465, MCS-83-05211 and U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027.

A Connectionist System for Rule Based Reasoning with Multi-Place Predicates and Variables

Lokendra Shastri and Venkat Ajjanagadde
Computer and Information Science Department
University of Pennsylvania
Philadelphia, PA 19104

Abstract

McCarthy has observed that the representational power of most connectionist systems is restricted to unary predicates applied to a fixed object. More recently, Fodor and Pylyshyn have made a sweeping claim that connectionist systems cannot incorporate systematicity and compositionality. These comments suggest that representing structured knowledge in a connectionist network and using this knowledge in a systematic way is considered difficult if not impossible. The work reported in this paper demonstrates that a connectionist system can not only represent structured knowledge and display systematic behavior, but it can also do so with extreme efficiency. The paper describes a connectionist system that can represent knowledge expressed as *rules* and *facts* involving *multi-place* predicates (i.e., *n-ary* relations), and draw *limited*, but *sound*, inferences based on this knowledge. The system is extremely efficient - in fact, optimal, as it draws conclusions in time proportional to the *length* of the proof. It is observed that representing and reasoning with structured knowledge requires a solution to the *variable binding* problem. A solution to this problem using a *multi-phase* clock is proposed. The solution allows the system to maintain and propagate an arbitrary number of variable bindings during the reasoning process. The work also identifies constraints on the structure of inferential dependencies and the nature of quantification in individual rules that are required for efficient reasoning. These constraints may eventually help in modelling the remarkable human ability of performing certain inferences with extreme efficiency.

1 Introduction

McCarthy in his commentary on Smolensky's paper: On the Proper Treatment of Connectionism [25] asserts that connectionist systems suffer from "the unary or even propositional fixation"; representational power of most connectionist systems is restricted to unary predicates applied to a fixed object. More recently, Fodor and Pylyshyn [12] have made sweeping claims that connectionist systems cannot incorporate systematicity and compositionality. These comments suggest that representing structured knowledge in a connectionist network and using this knowledge in a systematic way is considered difficult if not impossible. This paper addresses these concerns. It describes a connectionist system that can represent knowledge expressed in terms of $rules$ and $facts$ involving *multi-place* predicates (i.e., *n-ary relations*) and draw limited but sound inferences based on this knowledge in an extremely efficient manner. The time taken by the system to draw conclusions is proportional to the $length$ of the proof, and hence, optimal.

It is observed that the key technical problem that must be solved in order to represent and reason with structured and rule based knowledge is the *variable binding* problem [10, 26]. A solution to this problem using a *multi-phase* clock is proposed. The solution allows the system to maintain and propagate any number of variable bindings during the reasoning process.

1.1 A case for a strong notion of computational effectiveness

If we analyze human behavior we find that humans are extremely good at drawing certain kinds of inferences with remarkable efficiency - *often in a few hundred milliseconds*. We draw these inferences as if by $reflex$ and without deliberation. These inferences however, are by no means trivial and support a broad range of cognitive activity such as classifying and recognizing objects, understanding spoken and written language, and making commonsense inferences. Any artificially intelligent agent must match this remarkable performance in order to interact with intelligent agents and perform credibly in a complex environment. It therefore follows that any serious attempt at understanding intelligence must provide a detailed computational account of how such inferences may be drawn with requisite efficiency.

Researchers in knowledge representation and reasoning have been sensitive to the problem of tractable inference and have investigated several alternatives for computationally effective limited inference systems [13, 17, 18, 5, 19, 28]. The work described in this paper is also concerned with computational effectiveness but in a much stronger sense of the term, a sense suggested by the human reasoning capabilities mentioned above. Thus, we are not concerned here with questions of decidability or exponential, polynomial, or even linear complexity. What concerns us here is the search for appropriate knowledge structuring techniques and computational models that can be used to design systems capable of performing a limited class of inference with extreme efficiency - i.e., in time that is *at most sublinear* in the size of the knowledge base. An example of such efficiency would be a system that draws a class of inference in time proportional to the $length$ of the proof.

Unlike most of the work on knowledge representation where it is assumed that the target machine for the eventual implementation will be a serial von Neumann machine, this work presupposes a massively parallel (connectionist) architecture. It is our belief that working within a massively parallel computational architecture will help in identifying novel classes of limited inference that can be performed with extreme efficiency, and aid in discovering constraints that must be placed on the conceptual structure in order to support extreme efficiency [24]. Work described in this paper suggests that this belief is appropriate.

2 A Connectionist Metaphor for Reasoning

A connectionist system consists of a large number of highly interconnected but relatively simple processing elements. These elements communicate with their neighbors by propagating a level of activation and compute their own level of activity based upon the activation arriving from their neighbors.

Adopting a connectionist approach to knowledge representation and reasoning has some important consequences. We mention two that are relevant to the present discussion.

In a connectionist system there is no *distinct* interpreter that mediates retrieval and reasoning. The connection pattern, the weights on links, and the computational characteristics of nodes not only represent domain knowledge but also encode the retrieval and inferential processes that operate on this knowledge. This state of affairs *forces* a strong coupling between a representation and the inferences that the representation is expected to support. On the one hand, this makes the problem of representation more challenging, but on the other it directly addresses the central problem of efficient inference.[1].

The fine grain of parallelism supported by connectionism permits one to assign *a single processing element to each unit of information.* This has the following interesting consequence: Assume that besides enumerating facts about the world, we also identify the important *inferential connections* or *dependencies* between these facts. Now if we encode each piece of information as a connectionist node (henceforth node), and dependencies between pieces of information as explicit links between the appropriate nodes, then we can view inference as parallel spreading of activation in a connectionist network. The above metaphor has tremendous appeal because it suggests an extremely efficient way of performing inference. There are however, two critical problems that must be addressed before this metaphor can serve any purpose. These are the problems of convergence and control.

2.1 Convergence

Parallelism does not guarantee speed. In order to support extremely efficient inference, the spreading activation process must converge extremely fast. The computation performed by many connectionist systems corresponds to a *relaxation* process wherein activation circulates in a network until finally a stable network state is obtained. Often it is difficult to place an upper bound on the convergence time of such systems and even in cases where it is possible to do so, it often turns out to be polynomial in the size of the knowledge base. Given our emphasis on extreme efficiency, we wish to focus on systems that can perform desired inferences in time that is at most *sublinear* in the size of the knowledge base. This can be achieved by showing that the system is guaranteed to converge in a *constant* number - preferably one - of sweeps of spreading activation across the network. Such a convergence behavior ensures that the network can compute a solution in time proportional to the *diameter* of the network which is - in almost all cases - *sublinear* (and often *logarithmic*) in the size of the knowledge base. For example, in the context of inheritance and recognition in a semantic network, the diameter corresponds to the depth of (i.e., the number of levels in) the conceptual hierarchy, and is logarithmic in the number of concepts in the knowledge base.

But, for the connectionist network to compute solutions in a *single* pass of spreading activation the dependencies among pieces of knowledge must be *acyclic.* This condition may be satisfied naturally in some domains that exhibit restricted forms of cause-effect relationships much like open loop systems. In some other domains it may be possible to achieve this condition by carrying out a finer decomposition of the terms in the knowledge base so as to reduce

[1] For a detailed discussion of this issue refer to [22]

the density of dependencies (where density is the ratio of the number of dependencies to the number of terms in the knowledge base. A second way of eliminating cyclic dependencies is to identify suitable constraints on the conceptual structure that rule out certain types of cyclic dependencies.

2.2 Control of spreading activation, or the cross talk problem

By far the biggest snag in the spreading activation metaphor is that it overlooks the problem of control: It is one thing to build networks that model priming effects, simple associations, and associative recall, and another to build networks that can draw precise and controlled inferences. This problem is specially acute since connectionist systems operate without the intervention of an interpreter.

To appreciate the problem, consider a connectionist representation of a red square and a blue circle. We would expect this representation to be such that activating the representation of the red square would activate the representation of redness and squareness. Similarly, we would expect that activating the representations of blue circle would activate the representation of blueness and circleness. However, unless the representations and the rules of spreading activation are chosen carefully, the simultaneous activation of the representations of the red square and the blue circle will have the undesirable side effect of creating the representation of a red circle and a blue square! In the connectionist circles this problem is referred to as the *cross-talk* problem.

In order to avoid the cross-talk problem one needs a mechanism for *binding* the color value "red" with the representation of the red square and the color value "blue" with the representation of the blue circle and keeping these bindings from interfering with one another. A simple, almost trivial, way of achieving this is to posit the existence of a "conjunctive" node RED-SQUARE to represent the red square and have it connected to the nodes RED and SQUARE (which represent redness and squareness respectively). Similarly, one could posit the existence of a node BLUE-CIRCLE linked to the nodes BLUE and CIRCLE to represent the blue circle. [2]

The above scheme is only suitable for representing relatively stable (long term) knowledge. Such conjunctive nodes can clearly be learned over time in order to represent new - but stable - grouping of constituents. However, such a scheme is entirely inadequate if such groupings have to be created dynamically for short durations. Note that having a conjunctive node for all *possible* combinations of constituents is ruled out because such a solution requires too many nodes.

The need for establishing dynamic and temporary bindings clearly arises in language understanding, vision, and reasoning. In fact it arises in any situation that involves reasoning with representations that include the use of variables. [3] Consider the following example involving a simple reasoning step. Assume that a network encodes the rule:

$$\forall x, y\ [HIT(x, y) \Rightarrow HURT(y)]$$

and facts such as $HIT(John, Mary)$ and $HIT(Tom, John)$ among others. $HURT(John)$ clearly follows from the above knowledge by instantiating the rule with the bindings $John$ for y and applying modus ponens. If the network is to infer $HURT(John)$ it must carry out an *equivalent* computation. In generic terms, it must have a way of activating the representation of $HURT()$ given the activation of the representation of $HIT()$.

[2] An obvious problem with this solution is that it does not distinguish the roles of RED and SQUARE. In particular, the scheme does not allow RED-SQUARE to selectively activate RED without activating SQUARE. This is required because we would like the system to be capable of looking up the representation of the red square and answering the question "What is its color?". There are ways of solving this problem using relatively simple control mechanisms [23].

[3] Some researchers have argued that it may be possible to exhibit interesting cognitive behavior without solving the variable binding problem. For example see [1].

Furthermore, it must have a mechanism for establishing bindings for variables x and y in the representation of $HIT()$ and ensuring that the same bindings are induced in the representation of $HURT()$. The problem gets even more confounded if we wish to chain such inference steps and the bindings have to be propagated faithfully along the chain.

Note again that any solution that requires that such bindings be pre-wired is unacceptable: prewiring these bindings would correspond to explicitly representing all possible instantiations of the rule (this would also mean that the system is only dealing with propositions not with quantified sentences involving predicates!). This is not feasible because the number of instantiations may be too many - potentially unbounded. Thus we need the ability to set up these bindings on the fly.

3 Related Work

Before describing our system it may be appropriate to review some existing massively parallel systems for representation and reasoning.

Pearl's work[21] on "belief networks" involves the use of parallelism to perform efficient reasoning. These nets perform probabilistic reasoning where the fusion and propagation of beliefs proceeds in parallel provided the "causality" graph is *singly connected.* The system, however, only deals with propositions and does not allow variables.

Shastri [23] has described how hierarchically structured knowledge about concepts and their properties may be encoded as a connectionist network. The proposed connectionist semantic memory solves an interesting class of *inheritance* and *recognition* problems extremely fast - in time proportional to the depth of the conceptual hierarchy. The connectionist encoding is based on an evidential formalization of conceptual knowledge that leads to provably optimal solutions to the problems of *exceptions* and *multiple inheritance* during inheritance, and the *best match* or *partial match* computation during recognition. Shastri's system displays the desired level of efficiency as its response is at worst logarithmic in the size of the knowledge base. However, it does not address the problem of variable binding. For the sake of clarity if we suppress the evidential aspect of reasoning in the system, the system deals with rules of the form

$$\forall(x)P(x) \Rightarrow Q(x)$$

and

$$\forall(x)P(x) \Rightarrow Q(x, a)$$

Although multiple rules participate in a derivation, it is always the case that all variables are bound to the same individual and thus the system can get by without actually solving the variable binding problem.

The spreading activation metaphor described in Section 2 is not the only connectionist metaphor for reasoning. A second and quite different metaphor is the energy minimization metaphor [3] where the inference process is reduced to the problem of finding the lowest energy state(s) of a suitably interconnected network.[4] Such a process may even require not one but several cycles of convergence and it is difficult to place an upper bound on the convergence time of such systems. Even in cases where it is possible to do so, it turns out to be at best polynomial in the size of the knowledge base [6]. Thus, even though systems based on the energy metaphor are massively parallel - and are often explicitly motivated by a desire to build a system capable of performing certain inferences with extreme efficiency - they do not meet the efficiency requirement. Work described in [3, 27, 6, 7] belongs to

[4] The search for a minimal energy state is often carried out using a relaxation process such as simulated annealing [16].

this category. A second problem with such systems is that they are not always guaranteed to find the prescribed solution because the energy minimization process can get trapped in a local minima. In spite of these limitations we feel that work in this direction is quite significant in that it investigates an interesting alternative paradigm and leads to important insights.

Ballard and Hayes [3] were the first to develop a connectionist inference system using the energy minimization paradigm. They did not address the problem of variable binding as their system required that *all possible bindings be explicitly pre-wired* into the network. As we have argued above, such an assumption is much too strong.

Recently, Derthick[6] has proposed a connectionist system for drawing intuitively plausible inferences with respect to a frame based representation language that allows a limited use of quantifiers. The main strength of the system is its formulation of plausible inference that allows it to deal with conflicting information. As is the case with Ballard and Hayes' system, the main drawback of Derthick's system is that it lacks efficiency. The time complexity of the system turns out to be $polynomial$ in the number of terms (concepts, roles, fillers) in the knowledge base. Thus the system takes thousands of steps to solve even modest size problems.

Touretzky & Hinton [27] have described DCPS, a *distributed* connectionist encoding of a restricted production system. The operation of a production system requires the ability to perform variable bindings, and DCPS system exhibits this ability. The restrictions on variable bindings, however, are fairly strong. For example, DCPS only allows one variable in the antecedent. It also assumes that during any cycle there is only one rule with one variable binding that can constitute a potential correct match. Furthermore, it is possible that the system may not find the correct match. As DCPS is a production system, each step of the reasoning process involves rule selection (which in turn requires matching rule patterns with patterns in the working memory and selecting a winner) and updating of the working memory. Thus each step of the reasoning process involves (several) relaxation cycles, and therefore, the DCPS system does not satisfy the strong efficiency requirement. Recently, Dolan & Smolensky [8] have described a variant of the T&H production system using the tensor product representation proposed by Smolensky in [26]. The proposed system overcomes some of the limitations of the (T&H) system and its behavior is more amenable to analysis.

Dolan and Dyer[7] have proposed a system for schema selection and instantiation. The system uses a complex mechanism to bind schema roles (variables) to fillers (value). It appears that the role bindings are sensitive to the similarity of role fillers. Assume that filler1 is bound to role1 and filler2 has to be bound to role2. If filler2 is similar to filler1 then it is possible that filler2 may also get bound to role1. This is essentially a manifestation of the cross-talk problem discussed in Section 2.

4 Representation and Reasoning

The proposed connectionist system can perform a broad class of deductive inference involving variables and multi-place predicates with extreme efficiency. Specifically, the system can represent knowledge expressed in the form of $rules$ and $facts$ and determine whether a $query$ can be derived as a consequence of the facts and rules encoded in the system. The answers to queries are produced in optimal time: the time taken to draw an inference is only proportional to the $length$ of the proof.

The form of rules, facts, and queries is explained below.

Rules in the system are assumed to be sentences of the form

$$\forall x_1, ..., x_m \; [P_1(...) \wedge P_2(...) ... \wedge P_n \Rightarrow \forall y_1, ..., y_k \exists z_1, ...z_l \; Q(...)]$$

where arguments for P_i's are subsets of $\{x_1, x_2, ...x_m\}$, while the arguments of Q may consist of any number of arguments from among the x_i's and any number of constants besides the universally and existentially quantified arguments introduced in the consequent. Notice that the more commonly occurring rules of the form

$$\forall x_1, ..., x_m \; [P_1(...) \wedge P_2(...) ... \wedge P_n \Rightarrow Q(...)]$$

- where every variable occurring in a rule is universally quantified with the scope of quantification being the entire implication - are just a special case of the more general form specified above.

Facts are assumed to be atomic formulas of the form $P(t_1, t_2 ... t_k)$ where t_i's are either constants or existentially quantified variables.

A query has the same form as a fact: it is an atomic formula whose arguments are either bound to constants or are existentially quantified. Some examples of rules, facts, and queries follow:

Rules:

$\forall x, y, z \; give(x, y, z) \Rightarrow owns(y, z)$

$\forall x, y \; owns(x, y) \Rightarrow can\text{-}sell(x, y)$

$\forall x \; omnipresent(x) \Rightarrow \forall y, t \; present(x, y, t)$

$\forall x, y \; born(x, y) \Rightarrow \exists t \; present(x, y, t)$

$\forall x \; triangle(x) \Rightarrow number\text{-}of\text{-}sides(x, 3)$

$\forall x, y \; sibling(x, y) \wedge born\text{-}at\text{-}the\text{-}same\text{-}time(x, y) \Rightarrow twins(x, y)$

Facts:

$give(John, Mary, Book1)$; John gave Mary Book1.

$give(x, Susan, Ball2)$; Someone gave Susan Ball2.

$omnipresent(x)$; There exists someone who is omnipresent.

$triangle(A3)$; A3 is a triangle.

$sibling(Susan, Mary)$; Susan and Mary are siblings.

$born\text{-}at\text{-}the\text{-}same\text{-}time(Susan, Mary)$; Susan and Mary were born at the same time.

Queries:

1. $owns(Mary, Book1)$; Does Mary own Book1?
2. $owns(x, y)$; Does someone own something?
3. $can\text{-}sell(x, Ball2)$; Can someone sell Ball2?
4. $present(x, Northpole, 1/1/89)$; Is someone present at the north pole on 1/1/89?
5. $number\text{-}of\text{-}sides(A3, 4)$; Does A3 have 4 sides?
6. $can\text{-}sell(Mary, Ball2)$; Can Mary sell Ball2?
7. $twins(Susan, Mary)$; Are Susan and Mary twins.?

All queries except 5 and 6 follow from the rules and facts and the system responds 'yes' to these queries. The system says 'no' to queries 5 and 6.

4.1 Directed reasoning

The strong efficiency requirement we have imposed on our system entails that it find a solution in a fixed number of passes of spreading activation. As discussed in Section 1.2, this requires that the inferential dependencies in the knowledge base be acyclic. The nature of such inferential dependencies can be made explicit by expressing the

rule component of the knowledge base in the following graphical manner. Depict each predicate occurring in the rules by a unique node in the graph. Then if there is a rule of the form

$$P_1(...) \wedge P_2(...) ... \wedge P_n(...) \Rightarrow Q(...)$$

in the knowledge base, draw directed arcs from the nodes corresponding to P_is to the node corresponding to Q. The requirement that the inferential dependencies of the knowledge base be acyclic amounts to requiring that the directed graph obtained in this manner be acyclic. We will therefore focus on knowledge bases whose *inferential dependency* graph corresponds to a *directed acyclic graph* and henceforth, we will often refer to the rule component of the knowledge base as the PDAG (for Predicate DAG). Fig. 1 illustrates a PDAG corresponding to a small collection of rules.

In view of the directed nature of inferential dependencies, we refer to the system's inferential ability as *directed reasoning*. Directed reasoning appears to be adequate to capture a broad range of common sense reasoning situations. In particular, it can deal with restricted types of *causal reasoning*, i.e., reasoning about actions and events wherein there is no circular causality (i.e., systems that can be modeled as open loop systems). Terminological reasoning, that is, reasoning with definitional knowledge of concepts (terms) is also a case of directed reasoning.

5 The Connectionist Encoding

In this section we describe a connectionist system that can determine whether a query (i.e., an existentially quantified atomic formula) logically follows from a set of rules and facts. The soundness and completeness properties of this system are as stated in Section 8.

The proposed connectionist system only takes a *single pass* of spreading activation to answer queries, and while doing so the system maintains and propagates *any number* of variable bindings across *arbitrarily long* chains of inference.

5.1 An Overview

Before describing the encoding in full detail, we provide a brief sketch of how rules and facts are encoded and how queries are posed and processed by the network. Many details have been suppressed, subsequent sections provide a complete description.

In accordance with the connectionist metaphor for reasoning described in Section 2, we encode each "unit" of information by a distinct node and represent the inferential dependencies between these units by links between appropriate nodes. Fig. 2 depicts a network that encodes the following rules and facts:

Rules:

- $\forall x, y, z\ [orderhit(x, y, z) \Rightarrow hit(y, z)]$
- $\forall x, y\ [hit(x, y) \Rightarrow hurt(y)]$
- $\forall x, y\ [felldown(x) \Rightarrow hurt(x)]$

Facts:

- $felldown(bob)$

- $hit(dave, dick)$
- $hit(mike, dave)$
- $orderhit(bob, mike, dave)$

where the predicates may be interpreted as follows:

- $orderhit(x, y, z)$ - denoting that x ordered y to hit z.
- $hit(x, y)$ - denoting that x hit y.
- $felldown(x)$ - denoting that x fell down.

Each constant in the domain is represented by a *const* node (an oval shaped node). An *n-ary* predicate is represented by a *pred* node drawn as a rectangular box) and a cluster of n *arg* nodes (depicted as diamonds). Thus the ternary predicate *orderhit* is represented by the *pred* node labeled $ORDERHIT$ and the three *arg* nodes - $a1$, $a2$, and $a3$ - drawn next to it. As defined in Section 4, a fact is an instance of a predicate, some of whose arguments are bound to constants. [5] Each fact is represented by an *instancer* node (drawn as a hexagon) with two banks of inputs: one from the nodes representing the arguments of the corresponding predicate and the other from the nodes representing the constants that are bound to the arguments. The *instancer* node sends an output to the *pred* node. Thus the fact $orderhit(dave, mike, bob)$ is represented by the *instancer* node $A1$ connected to the *pred* node $ORDERHIT$, the *arg* nodes $a1, a2$, and $a3$ of $ORDERHIT$ and the *const* nodes *dave*, *mike*, and *bob* as shown in Fig. 2.

A rule is encoded by interconnecting the nodes representing the antecedent predicate and the nodes representing the consequent predicate. Thus the rule

$\forall x, y, z\ orderhit(x, y, z) \Rightarrow hit(y, z)$

is represented by connecting the nodes $ORDERHIT$ and HIT and the nodes representing the first and the second arguments of HIT ($a4$ and $a5$) to the nodes representing the second and third arguments of $ORDERHIT$ ($a2$ and $a3$) respectively.

Having described the basic representation, let us see how a query is posed to the network and the answer computed. Posing a query to the network involves specifying the query predicate and the bindings of the arguments in the query. To maintain and propagate the variable bindings, the system uses a *phased* clock, that is, a clock whose cycles are subdivided into several subcycles called *phases*.[6] The number of phases in a clock cycle equals the number of distinct bound arguments in the query, and can be ascertained when the query is posed. In particular, the number of phases does not depend on the length of the proof or the arity of the various predicates involved in the proof. While *pred* and *instancer* nodes remain active over entire clock cycles, *arg* and *const* nodes remain active only during those phases of a clock cycle in which they receive input activation.

The query predicate is indicated to the network by activating the corresponding *pred* node in the first clock cycle. Predicate nodes are insensitive to phases and therefore, the node corresponding to the query predicate may be activated during any phase of the first cycle. Each binding specified in the query is indicated to the network by activating simultaneously - i.e., in the same phase - the *arg* node and the *const* node corresponding to the binding. This process is repeated for each distinct binding - a different phase being used each time. During computation,

[5] In general, all arguments need not be bound.

[6] The use of the temporal dimension to disambiguate activity in connectionist networks has been suggested in the past [29, 9].

the simultaneous activation of an arg node and a $const$ node in a phase implies that the corresponding argument and constants are bound.

A query such as $hurt(dave)$ would require a clock with a single phase and would be posed to the network by activating the node $HURT$ (the node corresponding to the query predicate) in the first clock cycle, and the $const$ node $DAVE$ together with $a6$ - the appropriate arg node of $HURT$, in the first phase of the first clock cycle. This would result in the activation of the nodes as shown in Fig. 3. The pair of numbers next to the nodes indicate the clock cycle and the phase in which a node first becomes active. In brief, $a5$, the second argument of HIT, and $a7$, the first argument of $FELLDOWN$ become active in the first phase of the second clock cycle and thereafter remain active in the first phase of each subsequent clock cycle. As a result of coincident inputs from $a5$ and $DAVE$ during the first phase of each clock cycle, the $instancer$ node A2 would become active in the third clock cycle. Once activated, $A2$ would activate HIT in the fourth clock cycle which in turn would provide input to the query predicate $HURT$, indicating that the query is true. In general, the query predicate receiving input activation - other than the external activation that was provided when the query was posed - indicates that the query is true.

Beginning with the second clock cycle, the $instancer$ nodes $A3$ and $A4$ would receive activation from one of their arg nodes ($a5$ and $a7$, respectively) during the first phase of each clock cycle. However, as the associated $const$ nodes ($DICK$ and BOB, respectively) remain inactive during the phase the arg nodes are active, the $instancer$ nodes will not be activated.

The $instancer$ nodes play a key role in the reasoning process and the activation of an $instancer$ node indicates that the bindings of arguments in the fact encoded by the $instancer$ $subsume$ the bindings specified in the query.

The reasoning process amounts to a parallel backward chaining process. First the knowledge base is searched in parallel to locate all facts that satisfy the binding requirements imposed by the query and whose associated predicates have inferential connections to the query predicate. Next a parallel forward pass is made to determine whether the query follows from these selected facts. The flow of activation in the forward pass corresponds to the simultaneous progress of all possible proofs of the query formula. The whole computation is performed without any external intervention and in $optimal$ time - equal to twice the length of the proof.

5.2 Encoding constants, predicates, rules and facts

In this section we describe in detail the encoding of facts and rules. It turns out that the whole encoding can be described using nodes that behave like simple binary threshold units (BTU's) with phase sensitive behavior ($\rho - BTU$'s). An example of such a node is the arg node (see below) that becomes active in phase i of a clock cycle if it receives any input in phase i of the preceding clock cycle. Thus, we can view this node as a BTU - with threshold and input weights of 1 - whose sampling time window is a phase rather than a clock cycle. Another example of a phase sensitive variant of a BTU is the τ-or node (depicted as a triangle). A τ-or node remains active for a full clock cycle if it receives any input during one or more phase of the preceding clock cycle. A τ-or node can also be viewed as a BTU whose threshold as well as all inputs weights equal 1, except that its input time window is a phase while its output time window is a full clock cycle.

Even though the full system can be described using $\rho - BTU$s, in this section we will make use of two abstract node types, viz. $pred$ and $instancer$. We do so for clarity of exposition; the use of these abstract node types makes it easier to understand the encoding and its relation to the reasoning process. Later, in Section 7, we show how these nodes may be realized using $\rho - BTU$s. In describing the encoding we will also make extensive use of

inhibitory modifier links [11, 15] which have a simple computational interpretation. The activity along a link from a node a - after it is modified by an inhibitory modifier from a node b - can be interpreted as the signal $a\wedge \sim b$ (See Fig. 4).

A constant is encoded using a *const* node which is a simple phase sensitive BTU that becomes active in phase i of every clock cycle if it is initially activated in ith phase.

An *n-ary* predicate is represented by a *pred* node and an associated cluster of n arg nodes. Fig. 5 depicts the representation of a binary predicate $P1$. In all figures, *pred* and arg nodes will be depicted as rectangles and diamonds respectively. A *pred* node has three sites: IMP, INST, and BC. The role of the different sites of a *pred* node will be made explicit as we go along. As stated earlier, arg nodes become active in a phase i of a clock cycle if they receive one or more inputs in phase i of the previous clock cycle.

A rule is encoded by interconnecting the nodes representing the appropriate predicates. Fig. 6 depicts the representation of the rule

$\forall x, y, z P1(x, y, z) \Rightarrow Q1(y, x)$ ... $R1$

where $P1$ is a binary predicate and $Q1$ is a ternary predicate. The inferential connection between the predicates $P1$ and $Q1$ is encoded by a link from the *pred* node $P1$ to the site IMP of the *pred* node $Q1$ (IMP is an abbreviation of "implied"), and a link from $Q1$ to $P1$ incident at site BC (BC is an abbreviation for backward chaining). An input at the site IMP of a *pred* node means that the predicate is implied by the predicate from which it is receiving activation, while an input at site BC means that the predicate is of potential significance in establishing the truth of the predicate from which it is receiving activation.

The correspondence between arguments of antecedent and consequent predicates is established by links from arg nodes of the consequent predicate to the appropriate arg nodes of the antecedent predicate.

Fig. 7 provides additional details about how rules are encoded by illustrating the encoding of the rule:

$\forall x1, x2\ [P2(x1, x2) \Rightarrow \forall y \exists z\ Q2(x1, x2, y, z, a)]$ ..... $R2$

where $P2$ is a binary predicate and $Q2$ is a 5-ary predicate. The links between the first two arguments of $P2$ and $Q2$ reflect the argument mapping between the antecedent and the consequent. The triangular node labeled $g1$ is an τ-or node and it projects inhibitory modifiers that can block the spread of activation along links representing the inferential connections between the consequent and the antecedent of the rule R2. The significance of these inhibitory modifiers is as follows. If a variable that is existentially quantified in the consequent of a rule is bound to anything in the reasoning process then, it follows that this rule cannot be used to prove the consequent. This is reflected in the link from $a4$ to $g1$. If there is a constant in the consequent of a rule then, during the reasoning process the corresponding argument must not be bound to anything other than this constant. If this is not the case, the rule cannot be used to prove the consequent. This constraint is encoded by the link from $a5$ to $g1$ which is modified by an inhibitory modifier from the *const* node a. In most cases, the argument corresponding to the universal quantifier in the consequent has no direct bearing on the activation of the rule, and hence, need not be connected to anything. There is, however, one exception: if the same variable occurs in multiple argument positions in the consequent of a rule then we need to ensure that this variable is either unbound or bound to the same constant. This constraint is encoded by introducing a node that receives inputs from the arg nodes corresponding to such a variable. The encoding is illustrated in Fig. 8. The required node is shown as a pentagon and this node becomes active if it receives activation in more than one phase of a clock cycle.

As mentioned earlier, a *pred* node is an abstraction and can be realized using two simple BTU's. This realization is described in Section 7. The computational behavior of the abstract *pred* node is described in Fig. 9. A *pred* node has three states: Inert, Enabled, and Active; and three output levels: no output, low, and high corresponding

to each of the three states. A *pred* node changes state in the following manner: A low or high input at site BC causes a state change from Inert to Enabled. A low or high input at site INST or a high input at site IMP causes a state change from ENABLE to Active.

A fact is encoded using an *instancer* node. If a fact concerns a predicate of n arguments then the corresponding *instancer* node has $n+1$ sites: one ENABLE site and n BIND sites - one for each of the n arguments. At the ENABLE site an *instancer* receives an input from its associated *pred* node. At each BIND site an instance node receives an input from the appropriate *arg* node and if this argument is bound in the fact, then it also receives another input from the appropriate *const* node. The output of the *instancer* node goes to the site INST of the corresponding *pred* node. Fig. 10 depicts the encoding of the fact $P3(a,b,b,x)$, where $P3$ is a 4-place predicate with its fourth argument unbound.

An *instancer* node becomes active only if the bindings specified in the fact subsume the bindings specified in the query. This is achieved by ensuring that the *instancer* node becomes active if and only if in every phase, any BIND site receiving activation from its associated *arg* node also receives activation from its associated *const* node. Recall that each binding is specified by the simultaneous activation of the relevant *arg* and *const* nodes in the same phase of a clock cycle.

6 Inference Process

This section describes the inference process in more detail. For pedagogical reasons, we will begin by assuming that the system only consists of single antecedent rules. Subsequently, we will indicate how the system can be extended to deal with multiple antecedent rules.

The inference process may be thought of as consisting of three stages. [7] In the first stage, the query is posed to the network by external activation of some nodes. During the second stage, a controlled parallel search is carried out to locate all facts that are relevant to the proof of the query and the *instancer* nodes encoding such relevant facts become active. In the third and final stage the actual proof is constructed. In this stage, activation from the *instancers* denoting relevant facts flows downwards along the inference paths in the PDAG to produce an answer to the query. The answer corresponds to activation arriving at the *pred* node that corresponds to the query predicate along one or more of its ancestor *pred* nodes.

6.1 Posing the query and specifying role bindings

Posing the query involves specifying the query predicate and the constant argument bindings to the network. The query predicate is indicated to the network by activating the *pred* node that corresponds to the query predicate during the first clock cycle[8]. Bindings of arguments are indicated by using a phased clock. For a given query, each clock cycle of the network consists of a fixed number of phases. If the argument bindings in the query involve p distinct constants, then the clock has p distinct phases.[9] Let $c_1, ..., c_p$ be p distinct constants appearing in the bindings specified in the query. Then each clock cycle will have p phases. The query will be posed in the following manner:

[7] These stages are conceptually distinct, however, during actual processing these stages overlap.

[8] Predicate nodes are not phase sensitive, and they may be activated in any phase.

[9] In general p can be less than the number of bound arguments in the query because the same constant(s) may be bound to more than one argument.

In the i^{th} phase of the $first$ clock cycle, $(1 \leq i \leq p)$, the following nodes will be activated:

- The $const$ node corresponding to c_i.
- The arg nodes corresponding to the $i_1^{th}, ..., i_j^{th}$ arguments of the query predicate, where $i_1, ..., i_j$ are one or more arguments of the query predicate bound to c_i.

As stated earlier in Section 5.2, arg nodes and $const$ nodes are phase sensitive and the phases in which they remain active are determined by the clock phases in which they first become active. The simultaneous activation of an arg node and a $const$ node during a phase represents that the constant denoted by the latter node is bound to the argument denoted by the former node.

6.2 Searching for assertions

Once the query is posed, the second stage of reasoning ensues. During this stage, all assertions that are relevant to the proof of the query are identified. Relevant assertions can be of two types:

There may exist a fact associated with the query predicate itself whose argument bindings subsume the bindings specified in the query. The query would follow directly from such a fact. For example, the query $hit(dave, dick)$ (i.e., "Did Dave hit Dick?") trivially follows from the fact $hit(dave, dick)$ (refer to Fig. 2.). [10]

The second possibility is that there exist fact(s) associated with ancestor predicate(s) of the query predicate and whose argument bindings subsume those specified in the query. In this case, the query would follow via a chain of modus ponens.

We consider, in turn, how the two types of relevant facts become active during the query process. Consider how the $instancer$ node A2 (representing the fact $hit(dave, dick)$) becomes active in response to the query $hit(dave, dick)$. Once this query is posed, the $pred$ node hit remains active in every clock cycle, the $const$ node $dave$ and the first arg node of hit remains active during the first phase of every clock cycle. Similarly, the $const$ node $dick$ and the second arg node of hit remains active during the second phase of every clock cycle. The activation from these arg and $const$ nodes reaches the $instancer$ node A2 during the specified phases. $A2$ also starts receiving activation from hit beginning with the second clock cycle. An $instancer$ node functions as follows:

An $instancer$ node becomes active at the end of clock cycle t and remains active throughout cycle $t+1$ if and only if during the clock cycle t

- It receives activation from its associated $pred$ node, and
- During each phase of clock cycle t, if it receives activation from an arg node, it $also$ receives activation from the $const$ node bound to this arg node.

It follows that as a result of the query $hit(dave, dick)$, the $instancer$ A2 will become active at the end of the second clock cycle and remain active thereafter.

To see how relevant $instancer$ nodes associated with ancestors of the query predicate become active we shall consider an example. Consider the query $hit(mike, bob)$ (refer to Fig. 2). There is no fact associated with hit that subsumes the bindings in this query. As a result of the query, the first arg node of hit and the $const$ node $mike$ will become active in the first phase of every clock cycle. Similarly, the second arg node of hit and the $const$

[10] The fact $hit(dave, dick)$ also subsumes other queries such as $\exists x hit(x, dick)$, $\exists x hit(dave, x)$, etc., all of which also follow, directly, from this fact.

node *bob* will become active during the second phase of every clock cycle. Beginning with the second cycle and thereafter, the second *arg* node of *orderhit* will receive activation from the first *arg* node of *hit* during the first phase of each clock cycle, and the third *arg* node of *orderhit* will receive activation from the second *arg* nodes of *hit* during the second phase of each clock cycle.

As the phase in which an *arg* node becomes active depends on the phase in which they receive activation, the second and third *arg* nodes of *orderhit* become active in the first and second phases respectively of every clock cycle. Summarizing, the active *const* and *arg* nodes in the first phase of every clock cycle are: *mike*, the first *arg* node of *hit*, and the second *arg* node of *orderhit*; and those active in the second phase are: *bob*, the second *arg* node of *hit*, and the third *arg* node of *orderhit*.

Essentially, we have created two new bindings: *mike* has been bound to the second argument of *orderhit* and *bob* has been bound to the third argument of *orderhit*. At the same time the *pred* node *orderhit* will become active beginning with the second clock cycle as a result of receiving activation from *hit*. [11] The *instancer* $A1$ that encodes the fact $orderhit(dave, mike, bob)$ will now become active as a result of the activation it receives from *orderhit*, the second and third *arg* nodes of *orderhit* and the corresponding *const* nodes *mike* and *bob*.

What remains to be mentioned - as far as the search for facts relevant to the proof is concerned - is the role of the inhibitory connections to the links connecting the nodes that represent the antecedent and consequent predicates of rules (Fig. 7). As explained in Section 5.2, these inhibitory connections ensure that an implication between two predicates is used in a proof, only when some conditions implicit in the rule are met. The τ-or node, $g1$, at which these inhibitory links originate (refer to Fig. 7), checks whether the required conditions are met. If any condition is not met, the node $g1$ sends activation via inhibitory modifier links to block the flow of activation from the nodes that represent the consequent predicate to those that represent the antecedent predicate. Specifically, the inhibitory mechanism checks for three types of conditions.

The first of those conditions concerns the occurrence of constants in the consequent of a rule. Consider the rule

$$\forall x\ [P(x) \Rightarrow Q(x, a)]$$

where 'a' is a constant. This rule can be used in the proof of $Q(c, a)$ or $\exists y Q(c, y)$, but not of $Q(c, b)$. That is, if the second argument of Q is bound to a constant other than the one that appears in the rule, namely 'a', this rule cannot be used in a proof.

The second condition these inhibitory mechanisms check for concerns the existentially quantified variables that occur in the consequents of rules. Basically, it involves checking that these existentially quantified variables are not bound whenever the rule is made use of. As an example, given the rule

$$\forall x\ [P(x) \Rightarrow \exists y\ Q(x, y)]$$

and the fact $P(a)$, one can prove $\exists y\ Q(a, y)$, however, one cannot prove $Q(a, b)$.

The third type of condition checked by the inhibitory connections corresponds to the case where a variable occurs more than once in the consequent of a rule. This condition was not shown in Fig. 7. As an example, consider the rule

$$\forall x\ [P(x) \Rightarrow \forall y\ Q(x, x, y, a)]$$

This rule should not be used if the first and the second arguments of Q are bound in a query to different constants. they should be bound to the same constant. How this is achieved is illustrated in Fig. 8.

[11] The presence of these bindings together with the activation of the *arg* nodes of *orderhit* and *orderhit* can be thought of as encoding the query $orderhit(x, mike, bob)$ (i.e., "Did someone order Mike to hit Bob?")!

6.3 Producing the answer

At the end of the second stage, all the *instancer* nodes representing facts that are relevant to the proof of the query will become active. Once activated, each *instancer* will send activation to its associated *pred* node. Consequently, the *pred* nodes will switch to the Active state and transmit a high output (refer to Fig. 9, Section 5.2). This high output would impinge upon the IMP sites of all their child *pred* nodes which will in turn switch to an Active state and generate a high output. Eventually, activations originating at the active *instancer* nodes will flow downwards through the hierarchy of *pred* nodes to reach the query *pred* node which will then switch to an Active state - thereby producing an answer.

6.4 Reasoning with conjunctive antecedents

Previous subsections described the reasoning in the network when all the rules encoded in the network were single antecedent rules. Consequently, all the proofs were linear chains (note that though each proof is a linear chain, the search for a proof involves exploring all the branches of a potentially huge graph in parallel). When we allow rules of the form $P_1(...) \wedge P_2(...) \wedge ...P_m(...) \Rightarrow Q(...)$, the proof itself takes the form of a directed graph. Rules with conjunctive predicates in the antecedent are encoded using *conjunctive* nodes. The output of the *pred* nodes $P_1, ..., P_m$ are not connected directly to the *pred* node Q, instead they are connected a conjunctive node, which is in turn linked to the IMP site of the *pred* node Q. The output of the conjunctive node is high if and only if it receives activation through all the incoming links. The interconnections between the *arg* nodes of the antecedent predicates and the consequent predicate remain unchanged.

7 Realization of *Pred* and *Instancer* nodes

For ease of presentation we had introduced two abstract types of nodes, namely, *pred* nodes and *instancer* nodes to represent predicates and facts respectively. Like other node types used in the system, these abstract nodes are also built out of simple and neurally plausible units. This section describes the anatomy of these abstract nodes.

7.1 Encoding of *Pred* nodes using simple units

A *pred* node "subnetwork" consists of two units interconnected as shown in Fig. 11. We will refer to the triangle shaped node as *enabler* and the circular shaped node as *collector*. The links incident at the vertical edge of the *enabler* come from the child predicates of the predicate represented by *enabler* (these links are the ones shown as incident at the site BC of the *pred* node in the abstract description given earlier). The *enabler* unit is a simple *BTU* whose output becomes high if and only if it receives activation along one or more inputs (i.e., it computes an 'or' and can be realized as a *BTU* with weights and threshold of 1). An *enabler* node sends output to

- its associated *collector* node
- the *enabler* nodes of all the parent predicates
- the *instancers* associated with the predicate represented by the *enabler* node

The *collector* node receives two sets of inputs; one set of inputs comes from the associated *instancer* nodes and another set of inputs comes from the *collectors* of all the parent predicates. (these two sets of inputs correspond

respectively to those incident at the INST and IMP sites of the abstract *pred* node). The output of the *collector* node becomes high if and only if it receives activation from any of the parent predicates or any of the associated *instancers* (i.e., it is also an 'or' function over all its inputs). The *collector* node is also connected to the *collector* nodes of all the child predicates.

7.2 Encoding of Instancer nodes using simple units

Fig. 12 shows the details of an *instancer* subnetwork corresponding to an n-place predicate. 'P' is the input from the associated predicate node (i.e., from the *enabler* node representing the predicate). This input is modified by inputs from *arg* nodes. If an argument is bound to a constant in the fact represented by the *instancer*, the input from the corresponding *arg* node is in turn modified by an input from the appropriate *const* node. The τ-and node changes state only at the ends of clock cycles. It becomes active at the end of a clock cycle and remains active throughout the next clock cycle if and only if it receives activation during *all* the phases of that clock cycle.

7.3 Total node requirement

The total number of units required by the system is only *linear* in the number of predicates, rules, and facts. Each constant requires one *const* node. Each *n-ary* predicate requires two $\rho - BTU$ nodes and n *arg* nodes. Each fact requires one τ-and node. Each rule requires a node for every universally quantified variable introduced in the consequent that appears in more than one argument position in the consequent. Each rule also requires a τ-or node if there are any existentially quantified variables or constants in the antecedent. A rule with conjunctive antecedents also requires a *conjunctive* node. Thus the total node requirement is only linear in the size of the knowledge-base.

8 Soundness and completeness of reasoning

In this section we specify the soundness and completeness properties of the connctionist inference system system described in the preceding sections.

8.1 Single antecedent rules

We first focus on *single* antecedent rules, i.e., rules having the following general form:

$$\forall x_1, ..., x_m \; [P(...) \Rightarrow \forall y_1, ..., y_k \; \exists z_1, ...z_l \; Q(...)]$$

Single antecedent rules form an interesting subclass of rules. SA rules suffice to model a broad range of knowledge about actions and their effects. Notice that any rule of the form $\forall x_1, ..., x_m \; [P(...) \Rightarrow Q_1(...) \wedge ... Q_n(...)]$ can be decomposed into n rules of the form $\forall x_1, ..., x_m [P(...) \Rightarrow Q_j(...)]$. Thus even if an action P has multiple effects it may be possible to encode P using several single antecedent rules.

Another important sort of knowledge that can be modeled using single antecedent rules is terminological knowledge wherein complex terms (concepts) are described in terms of general terms until all terms get described with reference to a set of ground terms. [12] It is widely recognized that terminological knowledge is an important component of knowledge about any domain and leads to significant efficiencies in the reasoning behavior of knowledge

[12] As long as we avoid circular descriptions, terminological knowledge also satisfies the requirement that inferential dependencies induced by descriptions (rules) be acyclic.

representation systems [4]. Some of examples given in Section 4 can be viewed as examples of term descriptions. For example, the rule

$\forall x\ omnipresent(x) \Rightarrow \forall y,t\ present(x,y,t)$

can be viewed as a description of omnipresent, while the rule

$\forall x\ triangle(x) \Rightarrow polygon(x) \wedge number\text{-}of\text{-}sides(x,3)$

which when decomposed into two single antecedent rules

$\forall x\ triangle(x) \Rightarrow polygon(x)$

$\forall x\ triangle(x) \Rightarrow number\text{-}of\text{-}sides(x,3)$

can be viewed as the description of a triangle. It may be noted that permitting constants in the consequent allows us to express certain kinds of *role value* restrictions that are used in frame representation languages. It has been shown that the knowledge encoded in many semantic networks, frame based languages, or class/property systems can be expressed in terms of the following sorts of sentences both of which are special cases of single antecedent rules. [13]

$\forall x\ P(x) \Rightarrow Q(x)$

$\forall x\ P(x) \Rightarrow Q(x,a)$

It follows that the proposed system - even when restricted to single antecedent rules - can encode basic semantic network or frame-based representation languages. These and other issues pertaining to terminological reasoning are presented in detail in [2].

It can be shown that the proposed connectionist system is sound as long as the rules in the knowledge base obey the following condition:

Constraint-1: The same variable should not occur in more than one argument position in the antecedent of a rule.

If the above condition is satisfied the system will answer yes to a query only if the query is a consequence of the rules and facts encoded in the system. [14]

The condition stated above is in fact too strong, i.e., it is a sufficient condition and not a necessary one. We choose to state it as above because the specification involves only the *form* of the rules and does not refer to the nature of individual queries. It can be shown that our system - even though it may include rules that violate *Constraint-1* - will respond soundly to a query provided:

> any variable occurring in multiple argument positions in the antecedent of a rule that participates in the proof, gets bound as a result of the bindings specified in the query.

For example, the system's behavior will be sound even though it may include the following set of rules (notice that the first rule violates *Constraint-1*.)

$$\forall x,y\ P(x,x,y) \Rightarrow R(x,y)$$

$$\forall x,y\ R(x,y) \Rightarrow Q(y,x)$$

[13] The IS-A hierarchy can be encoded using the first kind of rules - for example, DOG IS-A ANIMAL can be expressed as $\forall x\ dog(x) \Rightarrow animal(x)$, and property value attachments may be encoded using the second kind of rules - for example, apples are red in color can be expressed as has-color(APPLE,RED).

[14] The consequent of a rule does not have to satisfy any such requirement.

provided the bindings specified in the query result in the first argument of R getting bound. Thus, queries such as $R(a,b)$ and $R(a,y)$ - where y is existentially quantified - as well as queries such as $Q(a,b)$ and $Q(y,a)$, will result in sound behavior but a query such as $R(x,b)$, $R(x,y)$, and $Q(a,y)$ *may not.* [15]

Like any limited inference system the proposed connectionist system is sound but incomplete. Below we identify the condition under which the system is complete, i.e., if this condition is satisfied, the system will always respond 'yes' to a query that follows from the facts and the rules encoded in the system. In order to describe this condition, we need to introduce some definitions.

Relevance: A predicate R is relevant to a query $Q(...)$ if

1. R is an ancestor of Q (w.r.t. to the PDAG)
2. a fact $R(...)$ is asserted in the system, and the bindings of $R(...)$ are such that it subsumes the query $Q(...)$
3. there is no predicate P between R and Q such that there is a fact $P(...)$ that subsumes the query.

Inference path: A path in the PDAG between two predicates.

Argument Mapping: An inference path between predicates P and Q induces a (symmetric) mapping between arguments of P and Q. This mapping is referred to as the argument mapping between P and Q.

Composite Argument Mapping: A composite argument mapping between P and Q is given by the *union* of the argument mappings between P and Q - the union being taken over all inference paths between P and Q.

Constraint-2: A condition for completeness

If the query logically follows from the rules and facts then the system will always answer yes provided there exists a relevant predicate R that satisfies *either* of the following conditions:

1. There is a unique inference path from the relevant predicate R to the query predicate Q.
2. The composite argument mapping between arguments of R and Q is such that any argument of R maps to at most one *bound* argument of Q and any bound argument of Q maps to at most one argument of R.

Notice that the existence of only one appropriate relevant predicate suffices. For example, given the following rules and facts:

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

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

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

$born(john, nyc)$

$moved(john, nyc, boston)$

$moved(tom, nyc, boston)$

the query $present(john, nyc, x)$ will succeed via the relevant predicate $born$ - even though the composite mapping between the other relevant predicate $move$ and the query predicate $present$ is such that two arguments of $move$ (second and third) map to the same argument (second) of $present$.

Also notice that only arguments bound in the query need to satisfy the condition stated in *Constraint-2* above. Thus a query such as $present(tom, x, y)$ will succeed via the relevant predicate $move$ because the argument that violates the condition (the second argument of $present$) is not bound in this query.

[15] Recall that variables in a query are assumed to be existentially quantified.

8.2 Conjunctive antecedent rules

The condition for soundness if we allow rules with conjunctive antecedents (i.e., rules of the form $P_1(...) \wedge P_2(...) \wedge ...P_m(...) \Rightarrow Q(...)$) is a simple generalization of the constraint *Constraint-1* proposed for the single antecedent case: For the system to draw sound inferences, the same variable should not occur more than once in the antecedent of a given rule (this covers all conjuncts of the antecedent).

As before, the above is a sufficient condition and the system will behave soundly with respect to a query if any variable occurring more than once in the antecedent of a rule that participates in the proof, gets bound as a result of the bindings specified in the query.

Similarly, the condition for completeness in the multiple antecedent case is a generalization of the single antecedent case. In order to make this generalization explicit, consider the "proofs" of a query in the single antecedent case and the multiple antecedent case. In the single antecedent case, a proof corresponds to a $path$ in the PDAG from the relevant predicate to the query predicate. In the multiple antecedent case, however, a proof corresponds to a directed *graph* whose root node is the query predicate and whose tip nodes correspond to relevant predicates. Call this graph a *solution graph.* [16] The completeness condition in the multiple antecedent case is as follows:

> The system will always answer 'yes' to a query that logically follows from the rules and facts if there exists at least one proof such that every relevant predicate (tip node) in the corresponding solution graph satisfies *Constraint-2* - the requirement for completeness stated in Section 4.3.

9 Conclusion

This report describes a connectionist system for performing a restricted class of deductive reasoning using multi-place predicates and variables in an extremely efficient manner. The time taken by the system is proportional to the length of the proof, and hence, optimal.

The proposed system is very powerful yet very limited. So although it can draw certain kinds of inferences in optimal time while maintaining and propagating variable bindings, it also places constraints on the structure of inferential dependencies and the nature of quantification in the rules. These mixed attributes have to be interpreted in the correct context. It is important to recognize that we did not set out to building an efficient $PROLOG$ engine or an efficient *general purpose* production system. Instead, we wished to arrive at a detailed computational account of how certain restricted but nevertheless complex inferences can be performed almost effortlessly, almost as if they were a $reflex$ response.

What emerges quite clearly is that in order to support extreme efficiency, inferential dependencies between pieces of knowledge must be represented explicitly and vividly. Thus every rule and fact in the system occupies a precise place in the knowledge structure. Adding a rule is not difficult as the rules for inserting a rule are precisely known and can be automated. However, learning a new rule appears to be more involved. But it must be remembered that in our context 'learning a rule' means making the rule an integral part of the agent's conceptual knowledge so that this rule can henceforth be used by the agent almost as a matter of reflex without conscious thought. Clearly, none would argue that learning new rules - in this sense - is easy for humans.

[16] In the multiple antecedent case, the PDAG can be viewed as an AND/OR graph [20]: Each set of rules having the same consequent predicate introduces an OR node, and every rule with multiple antecedents introduces an AND node. The solution graph mentioned above can be thought of as a solution graph of the AND/OR graph corresponding to the PDAG.

In the near future we will report an augmented that can answer *wh*-questions in addition to 'yes/no' questions (i.e., the augmented system is capable of determining the fillers of unbound arguments in the query). We will also show that there exists a direct way of integrating a connectionist semantic network (i.e., an inheritance network) such as the one described in [24] and the rule-based system described here. Such a 'hybrid' system will have more expressive and inferential power but will retain its extreme efficiency.

The use of time to represent variable bindings has wide applications - some of these are being investigated [2]. We are also looking at probabilistic generalizations of the system described here wherein rules can be viewed as preference rules.

10 Acknowledgements

We wish to thank the Knowledge Representation group at the International Computer Science Institute, Berkeley, in particular, Jerry Feldman and Mark Fanty for their helpful comments and suggestions.

References

[1] Agre P.E. and Chapman, D. Indexicality and the Binding Problem. Presented at the Symposium "How Can Slow Components Think So Fast". Stanford University, Palo Alto, CA. 1988.

[2] Ajjanagadde V. Forthcoming Ph.D. dissertation, University of Pennsylvania.

[3] Ballard, D.H., and Hayes, P.J., Parallel logical inference, Proceedings of the Sixth Annual Conference of the Cognitive Science Society, pp. 114-123., Boulder,Colorado, June.1984.

[4] Brachman, R., Fikes R., and Levesque, H.J. KRYPTON: A Functional Approach to Knowledge Representation. *Readings in Knowledge Representation*, R. Brachman, and H.J. Levesque (eds.) Morgan Kaufman, Los Altos, CA. 1985.

[5] Brachman, R. and Levesque, H.J. The Tractability of Subsumption in Frame-Based Description Languages, *AAAI*-84.

[6] Derthick, M., Mundane reasoning by parallel constraint satisfaction, Ph.D. thesis, CMU-CS-88-182, Carnegie Mellon University, Sept. 1988.

[7] Dolan, C., and Dyer, M., Parallel retrieval and application of conceptual knowledge, Technical Report TR UCLA-AI-88-3, University of California, Los Angeles, Jan. 1988.

[8] Dolan, C., and Smolensky P., Implementing a connectionist production system using tensor products Technical Report UCLA-AI-88-15, University of California, Los Angeles, CU-CS-411-88 University of Colorado, 1988.

[9] Fanty, M.A., Learning in Structured Connectionist Networks. Ph.D. Dissertation, Computer Science Department, University of Rochester, Rochester, NY. 1988.

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

[11] Feldman, J.A. and Ballard D.H. Connectionist models and their properties. *Cognitive Science*, 6 (3): 205-254, 1982.

[12] Fodor J.A. and Pylyshyn Z.W. Connectionism and cognitive architecture: A critical analysis. In *Connections and Symbols* Steven Pinker and Jacques Mehler (eds.) The MIT Press, Cambridge, MA. 1988.

[13] Frisch, A.M. and J.F. Allen, Knowledge retrieval as limited inference in D.W. Loveland(Ed.), Lecture Notes in Computer Science: 6th Conference on Automated Deduction, Springer-Verlag, New York, 1982.

[14] Hinton, G.E. and T.J. Sejnowski, Optimal perceptual inference, Proc. IEEE Computer Vision and Pattern recognition Conference, pp. 448-453, Washington, DC, 1983.

[15] Kandel, E.R. *The Cellular Basis of Behavior*. Freeman, San Francisco, CA. 1976.

[16] Kirkpatrick, S., C.D.Gelatt, an M.P. Vecchi, Optimization by simulated annealing, *Science* 220, 4598, pp. 671-680, 1983.

[17] Levesque, H.J., A logic of implicit and explicit belief, *AAAI*-84, pp. 198-202.

[18] Levesque, H. and Brachman, R., A fundamental tradeoff in knowledge representation and reasoning in *Readings in Knowledge Representation*, R. Brachman and H. Levesque (eds.), Morgan Kaufman, 1985.

[19] McAllester, D., An outlook on truth maintenance, AI memo 551, MIT AI Lab., 1980.

[20] Nilsson, N.J., *Principles of Artificial Intelligence*. Tioga Publishing Company, Palo Alto, CA. 1980.

[21] Pearl, J., Fusion, propagation and structuring in belief networks, *Artificial Intelligence*, 29(3), 1986, pp. 241-288.

[22] Shastri, L., The Relevance of Connectionism to AI: A representation and reasoning perspective. In *Advances in Connectionist and Neural Computation Theory*, vol. 1., J. Barnden (ed.), Ablex Publishing Company, Norwood, N.J. (To appear). (Also available as a Tech. Report from Computer Science Department, University of Pennsylvania.)

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

[24] Shastri, L., A connectionist approach to knowledge representation and limited inference, *Cognitive Science*, 12(3), pp. 331-392.

[25] Smolensky, P., Proper treatment of Connectionism, *Behavioral and Brain Sciences*, (1988) 11:1.

[26] 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, Feb. 1987.

[27] Touretzky, D. and Hinton, G., Symbols among neurons: Details of a connectionist inference architecture, Proceedings of $IJCAI$-85 pp. 238-243.

[28] Vilain, M., The restricted language architecture of a hybrid representation system, Proceedings of $IJCAI$-85, pp. 547-551.

[29] von der Malsburg, C., Nervous structures with dynamical links, *Berichte der Bunsen-Gelschaft fur Physikakalische Chemie.*

RULES

W(...) & U(...) -> X(...)

P(...) & Q(...) & R(...) -> T(...)

T(...) & S(...) -> U(...)

U(...) -> V(...)

M(...) -> X(...)

PDAG

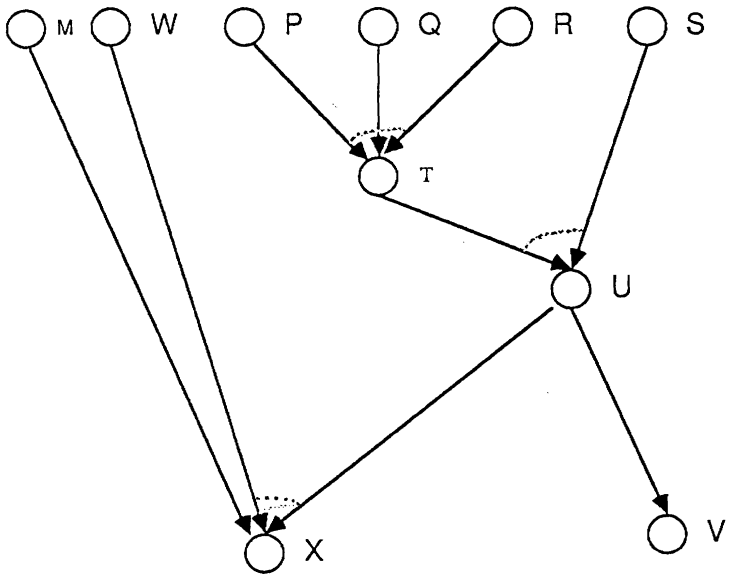
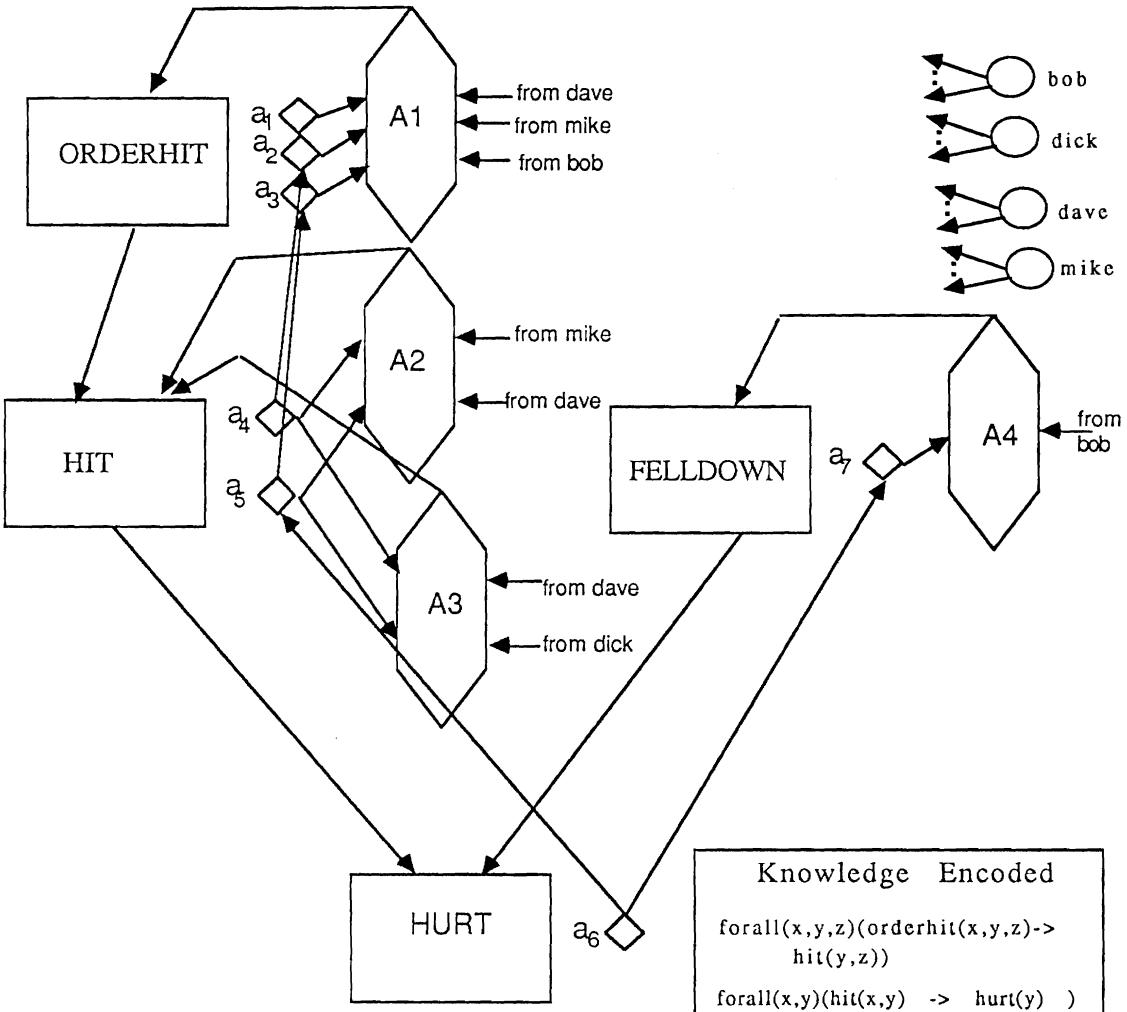


Fig. 1 PDAG For An Example Set Of Rules



Knowledge Encoded

forall(x,y,z)(orderhit(x,y,z)-> hit(y,z))

forall(x,y)(hit(x,y) -> hurt(y))

forall(x)(felldown(x) -> hurt(x))

orderhit(bob,mike,dave)

hit(mike,dave)

hit(dave,dick)

felldown(bob)

Fig.2 An Example Network

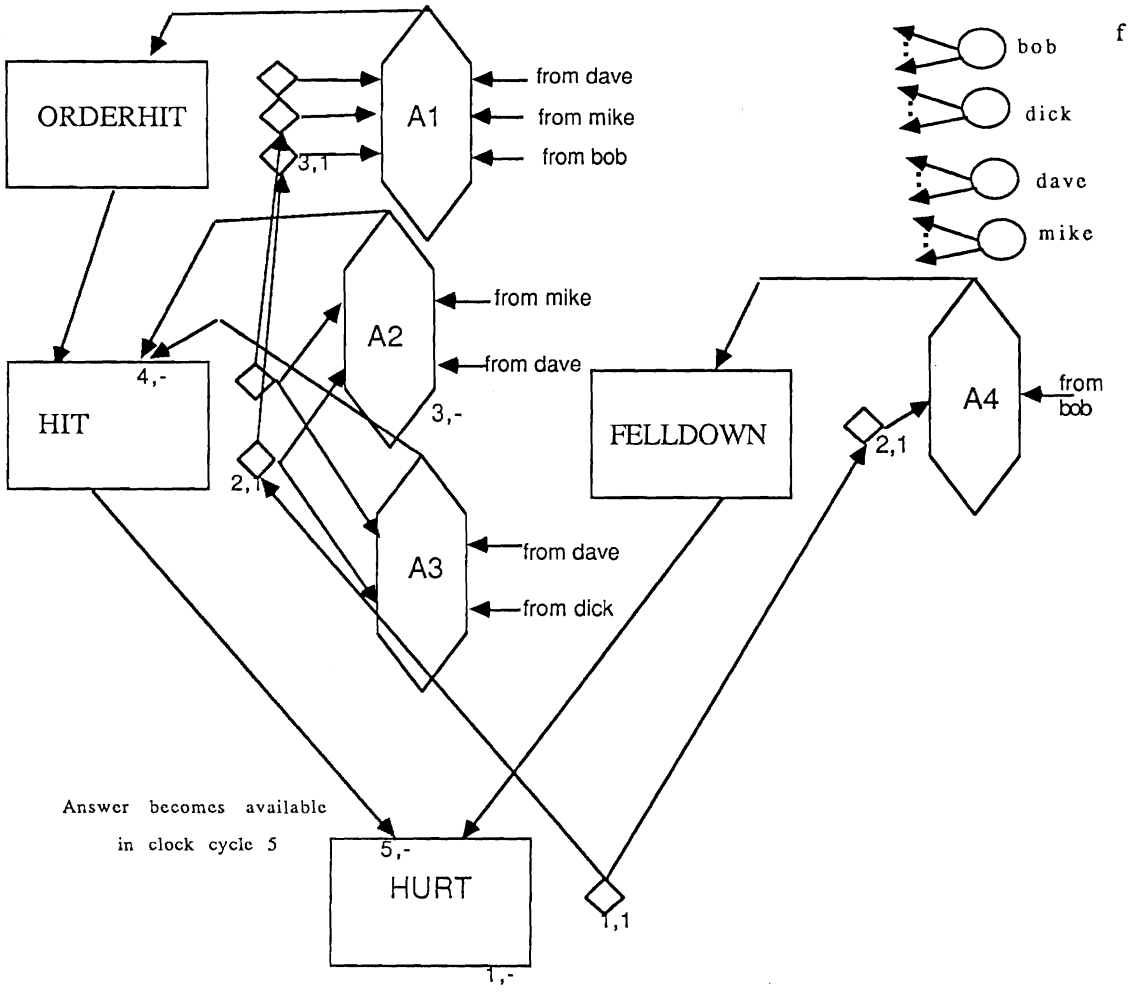


Fig. 3 Timings for the query Hurt(dave)

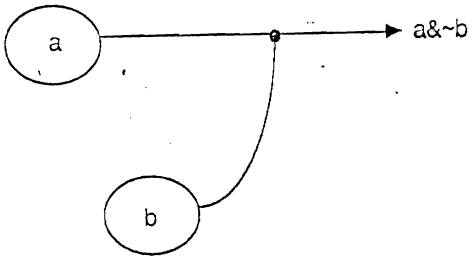


Fig. 4 Modifier Link

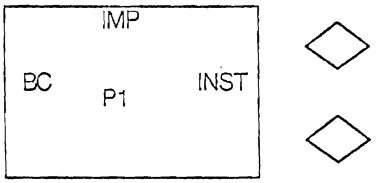


Fig. 5 Representation of a binary predicate $P1$

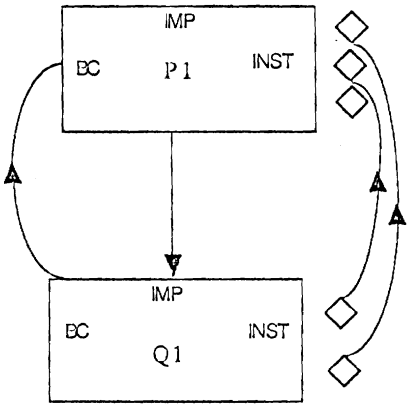


Fig. 6 $\forall x, y, z\ P1(x, y, z) \Rightarrow Q1(y, x)$

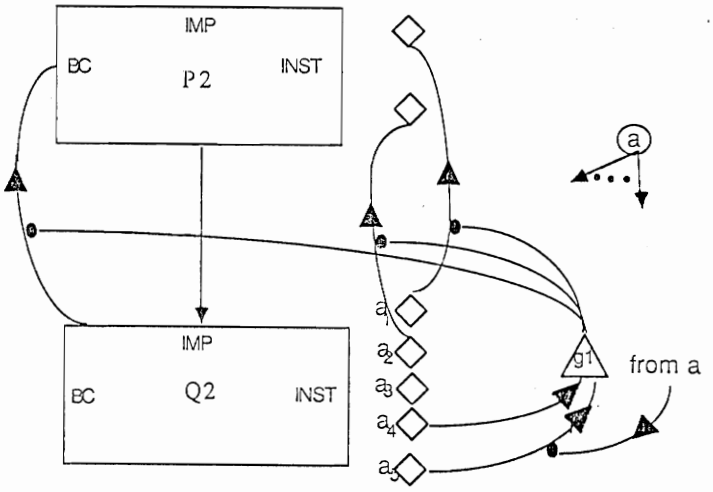


Fig. 7 $\forall x1, x2\ [P2(x1, x2) \Rightarrow \forall y \exists z\ Q2(x1, x2, y, z, a)]$

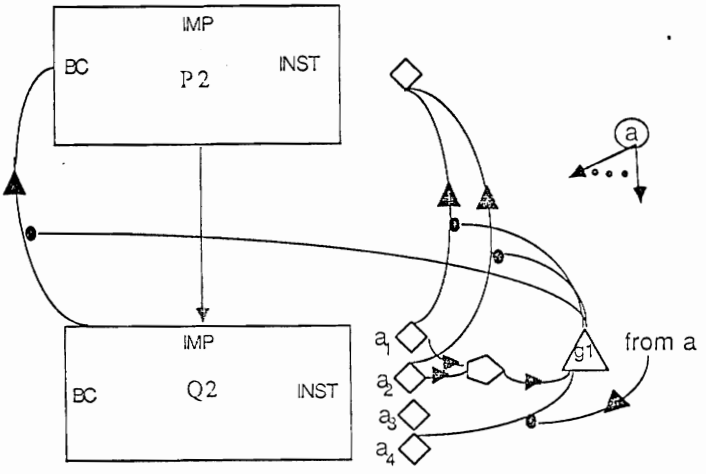


Fig. 8 $\forall x1\ [P2(x1) \Rightarrow \forall y1\ Q2(x1, x1, y1, a)]$

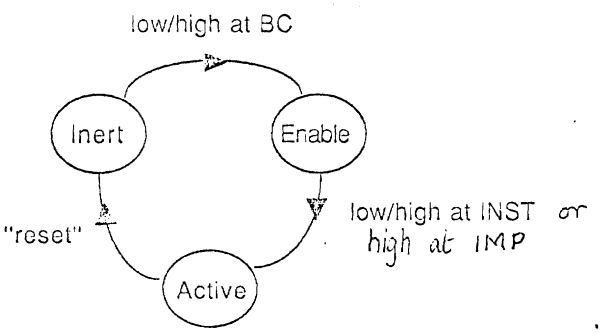


Fig. 9 Computational Behavior of the Abstract Pred Node

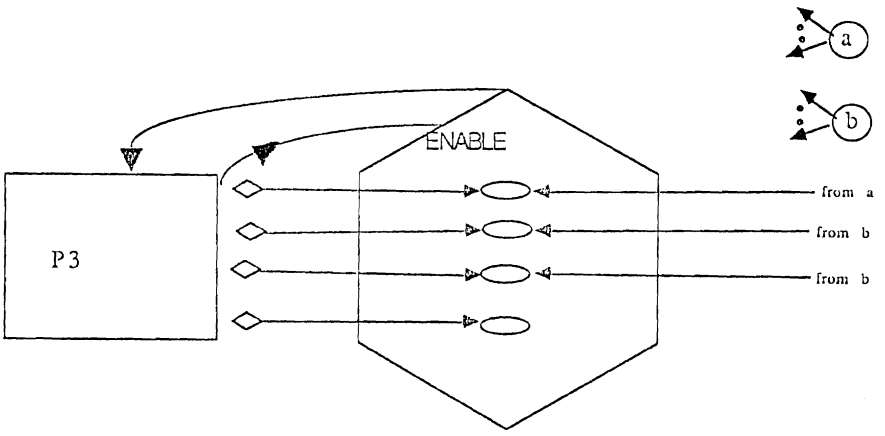


Fig. 10 Abstract Instancer Node for $\exists x$ (P3 (a,b,b,x))
(Ellipses denote bind-sites)

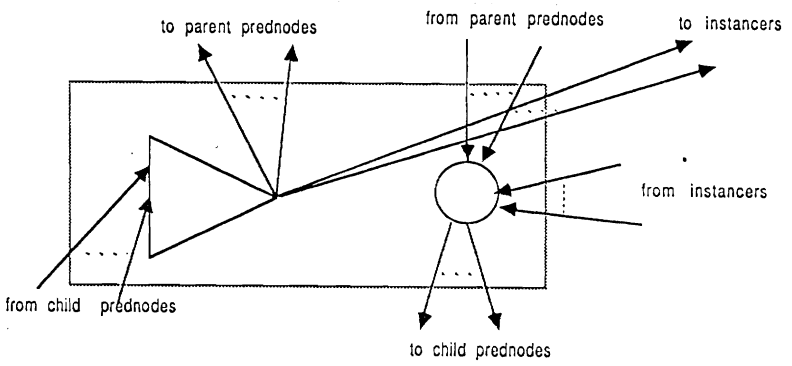


Fig. 11 Realization of a Pred Node

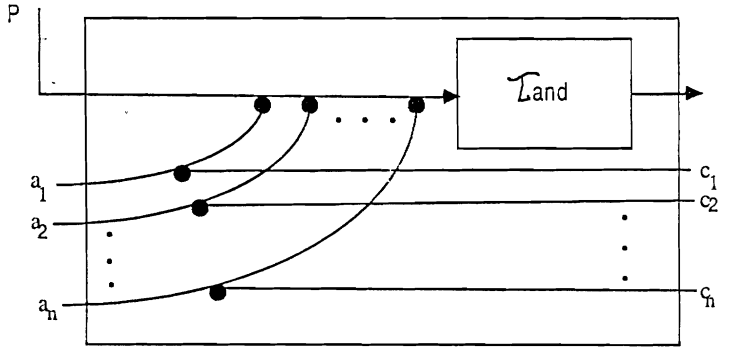


Fig. 12 Realization of an Instancer Node