Dynamics of Knowledge Processing in Parallel Systems

Abhijit Chaundhury

Dan C. Marinescu

Andrew Whinston

Report Number: 88-776

Chaundhury, Abhijit; Marinescu, Dan C.; and Whinston, Andrew, "Dynamics of Knowledge Processing in Parallel Systems" (1988). Department of Computer Science Technical Reports. Paper 665.

https://docs.lib.purdue.edu/cstech/665

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
DYNAMICS OF KNOWLEDGE PROCESSING IN PARALLEL SYSTEMS

Abhijit Chaudhury
Dan C. Marinescu
Andrew Whinston

CSD TR-776
May 1988
DYNAMICS OF KNOWLEDGE PROCESSING 
IN PARALLEL SYSTEMS

Abhijit Chaudhury 
Krannert School of Management 
Purdue University 
West Lafayette, Indiana 47907

Dan C. Marinescu 
Department of Computer Sciences 
Purdue University 
West Lafayette, Indiana 47907

Andrew Whinston 
Department of Computer Sciences and 
Krannert School of Management 
Purdue University 
West Lafayette, Indiana 47907

Abstract

Inferential reasoning systems such as Prolog or expert systems are treated as computational systems in the framework of net theory. Once the representation is established the collection of analysis techniques associated with net theory can be applied to a wide range of artificial intelligence (AI) systems. In the paper we exploit these techniques to explore issues of inconsistency and contradictions in the knowledge base, detect deadlock, and recognize redundancy. One major advantage of net theory is the possibility of analyzing parallelism in the inferential process. We specify a formal model that maps the computational model in its net representation to an appropriate parallel architecture. Net theory is also used to model extensions to horn-clause systems such as belief structures and non-horn clausal systems. It appears that the representation of inferential process in net theory is a useful tool in that many of the well developed techniques of analyses can be applied.
1. INTRODUCTION

In this study we construct a class of computational models for various types of inferential reasoning systems, based upon a net theoretical approach. We believe strongly that a net theoretical viewpoint could lead to a unified framework for: (a) specification of knowledge processing systems, (b) validation and analysis of the resulting computational models, (c) mapping of the models to parallel architecture, and (d) performance analysis of knowledge processing systems.

Different models have been used successfully for representation of knowledge processing systems. The use of directed and acyclic networks as a syntactic device for representing facts in a first-order-logic systems, has been known in the AI literature [33]. Directed networks have also been utilized to represent belief-networks and probabilistic dependencies [35]. In the specific area of modeling logical systems net models have been employed to represent first order propositional logic [41], modal logic [12], and predicate calculus [16]. However, none of the existing models seems powerful enough to capture the dynamics of knowledge processing, to the extent that algorithms which map a computational model to a parallel system could be constructed.

We will be adopting an interdisciplinary point of view, employing concepts from graph theory and net theory, which will turn out to be fruitful for suggestion of parallel algorithms that can be implemented on multi-processor architectures. Net theory is a theory of system organization whose origin is the seminal dissertation of Carl Adam Petri in 1957 [37]. This work has triggered an extremely rich body of research, covering all major areas of concurrent distributed systems. A 1987 bibliography of the field includes more than 2000 titles. The Petri nets as computational models have been shown to be equivalent to Turing machines. They have been applied successfully to various areas of computer science, ranging from Computer Organization, Specification and Validation of Protocols and Data Bases to Performance Models, Software Engineering, Productions Systems and Office Automation.

The life cycle of any computing system consists of a set of distinct but closely related activities. The cycle starts with system specification at a level of abstraction suitable for machine representation. Net theory has been used for this purpose in some of the areas mentioned above. For example, net models have been used for specification of real-time systems. The next element of the life cycle is the analysis of the system specifications, to establish whether the systems described by the specifications correspond to the design goals, have all the desired properties. The main strength of the net models in all the application areas mentioned above is precisely in the area of system analysis. The performance analysis is another important aspect of any system. Timed nets have been used successfully for this purpose, in areas like multiprocessor performance analysis, performance analysis of communication protocols, etc.

The question we explore now is, what are the relevant properties of the net models which make them powerful enough to support the four objectives mentioned above in connection with knowledge processing systems?

As far as the representation is concerned, we have already indicated the results from [45], which show that the net models are as powerful as Turing machines. Specification of a system as a net has further advantages due to the fact that the two dimensional syntax is suitable for visualization. This has two important consequences: from one point of view models of complex interacting systems become readable, and on the other hand it changes the environment in which the system designer operates. Rather than writing large programs using a specialized programming language the designer will use a graphics editor and a library of graphics sub-models. An automated tool will translate the graphic models into executable programs, or will generate code in LISP, OPS5 or another suitable language. This allows the human designer to concentrate on the functionality of the system, since he deals with a higher level of abstraction, rather than on
implementational details. The higher level of abstraction makes the system description machine independent.

Let us now examine briefly how the validation and analysis of computational models can be carried out in the framework of net theory. The net systems capture both the static and the dynamic characteristics of systems they represent. The static properties are captured in the structure of the underlying net. The dynamic properties are related to the flow of entities through the system.

For knowledge processing systems, the net analysis can be used to discover inconsistencies and contradictions in the knowledge base, to detect deadlock, to study redundancy, etc. Net theory provides three classes of methods for the analysis of the behavior of a net system:

- the study of the reachability set, the set of all states of the system,
- transformation methods based upon net morphisms, and
- the study of the invariants. For complex systems with large state spaces, the method of invariants is extremely powerful since it allows us to investigate particular subnets with desirable properties.

The aspect of the net system which has received the least attention seems to be related to the mapping of the computational models to parallel architectures. The traditional net models are timeless and though they model very clearly some of the fundamental aspects of concurrent processing, they are not suited for this task. For this reason, we augment our net models with timed transitions. Since knowledge processing is computationally intensive, knowledge processing systems may have to be implemented on massively parallel systems. A vast body of literature is devoted to mapping of parallel computations to parallel machines. Most of the algorithms known to us are based upon one form or another of computational graphs with nodes describing the computational and communication requirements of concurrent processes.

Net models are essentially flow models, which make explicit the higher level of parallelism possible in a given computation. In the net model, we view a computation as a transition which consumes data in its input places and produces results in its output places. To map such a model to a parallel machine means to aggregate groups of places and transitions into subnets and then assign these subnets for processing on the set of available processors. This aggregation conceivably could be done at run time by an intelligent scheduler subject to optimization constraints. Using a synchronic distance metric to be described later, we hope to outline formally the concurrency aspects of this aggregation. A side effect of this approach with possible significant consequences, is the possibility of mapping AI systems into distributed architectures.

Last but not least, the net models will be used for the performance analysis of the systems. Techniques for these tasks are now available and several results concerning performance analysis of different types of parallel systems have been published [31].

In this paper we introduce colored propositional nets based upon colored nets as defined by Jensen [23]. They will be used for specification and modeling of propositional logic, predicate calculus, and production systems. While the idea of using a net representation for AI systems is not new, (see for example the modeling of expert systems by Predicate Transition nets [16]), this paper will present a systematic view of the field with a vastly more expanded objective relating to parallel processing, knowledge representation, model validation, etc.

To make this paper self-contained we provide in Section II an introduction to net theory. We identify the major components of the theory and describe methods of *invariants* and *reachability* trees for studying the behavior of the modeled system. We furnish definitions and instances of various types of interrelationships possible among informational processes viz., *concurrency, conflict and confusion*, and elaborate on the idea of synchronic distances which is
a measure of dependency among events occurring in a distributed system. There are several net models called high level nets such as colored net and predicate transition nets that permit parsimonious representation of large systems. We describe in brief the relevant high level models in this section.

In the next two sections, we show how horn clauses and their generalization can be represented in our system. We lay out the formal net models that we will be employing in this paper. We describe parallel algorithms for doing forward inference in each of the above systems and discuss their computational properties.

In Section V, we demonstrate a colored net representation of a typical Expert system. We will use net representation to observe the system at different levels of detail. Net theory will help us to describe in a formal manner, parallelism inherent in the system at different levels of abstraction. In Section VI, we will discuss the mapping problem, i.e., the problem of mapping a given algorithm onto a given multi-processor architecture. This will be done in the context of a net representation of an expert system. A graph-theoretic definition of the mapping problem will be furnished. Net theory will help us to define several metrics that will be able to describe, succinctly, an expert system algorithm from the point of view of parallelism and how well it relates to any given multiprocessor architecture. The problem of allocation of rules of an expert system to individual processors will be studied and a quadratic assignment problem will be formulated, which will attempt to maximize concurrency while limiting communication overhead. The assignment problem will be shown to be NP-complete and heuristic methods for solution will be offered.

Section VII will identify several knowledge engineering issues and show how our formalism provides a basis for providing software assistance to knowledge engineers. Finally, in Section VIII, we summarize our research and provide suggestions for future development.

II. NET THEORY

This section provides an overview of selected concepts from net theory necessary for the understanding of computational models proposed in this paper.

We discuss briefly, three types of nets: Elementary Net Systems (EN), which represent the basic net models powerful enough to represent the fundamental concepts of concurrent systems, and more sophisticated nets, such as the Place Transition Net (PT), and the High Level Nets. For an in-depth coverage of these topics, the reader is referred to Thiagarajan [44] and Rosenburg [41] for EN, to Reisig [40] for PT nets, and Jensen [4] for Colored Petri nets. A survey of applications can be found in [48]. The notations and the concepts from these papers are followed closely in our presentation.

II.A Elementary Net Systems

A theory of distributed systems is characterized by how it chooses to formulate the basic notions of 'states' and 'changes-of-states'. The points of view adopted by net theory are (i) states and transitions (changes-of-states) are two intertwined but distinct notions that deserve even-handed treatment, (ii) both states and transitions are distributed entities, and (iii) the extent of a change caused by a transition is fixed, it does not depend on the state in which it occurs. A transition is enabled to occur at a state if and only if the fixed extent of change associated with the state is possible at that state. The basic concepts of the theory of Elementary Net Systems are defined in the following.

Definition 1. (Net)
A directed net is a triple \( N = (S, T; F) \) where

1.3 \( S \cap T = \emptyset \)
1.2 \( S \cup T = N \)
1.3 \( F \) a subset of \((S \times T) \cup (T \times S)\), where \( F \) is called the flow relation and its elements are referred to as arcs.
1.4 \( \text{dom}(F) \cup \text{codom}(F) = S \cup T \)

**Definition 2.** (Preset, Postset)

Let \( N = (S, T; F) \) be a net and \( x \in X = S \cup T \), then

\[
\begin{align*}
*x &= \{ y \mid yFx \} \quad \text{(the preset of } x) \\
*x &= \{ y \mid xFy \} \quad \text{(the postset of } x).
\end{align*}
\]

**Remarks**

1. When a system is represented by a net, the \( S \)-elements correspond to local states, the \( T \)-elements to local transitions, and the flow relations to neighbor relations between local states and local transitions.
2. In the basic net models called Condition Event Nets (C/E), the \( S \)-elements represent conditions and the \( T \)-elements represent events. Such a net is denoted by \( N = (B, E; F) \).
3. A case of a C/E net is a subset of conditions \( c \subset B \). In a diagram, a case \( c \) is represented by marking the conditions which are members of \( c \).
4. Given the event \( e \in E \), \( *e \) is the set of preconditions and \( e* \) is the set of postconditions, of the event \( e \).
5. An event occurs at case \( c \) if all preconditions hold and none of the postconditions hold. Formally, we express this as \((*e \subset c) \land (e* \cap c = \emptyset)\). The event \( e \) is now said to have a concession in case \( c \).

**Definition 3.** (EN System)

3.1 An Elementary Net system (EN), is a quadruple \( N = (B, E; F; c_{in}) \) with \( (B, E; F) \) the underlying net, and \( c_{in} \subset B \) the initial case of \( N \).
3.2 An EN system is contact free iff \( \forall e \in E \) and \( \forall c \in C \) then \(*e \subset C \Rightarrow e* \cap c = \emptyset\).

**Remarks**

1. EN systems satisfy the general principles for system representation mentioned at the beginning of this section. The states of the system are represented by sets of conditions and the changes of state by a set of events.
2. If a system is contact-free, it is sufficient that the precondition of an event hold at a case \( c \) in order that the event occurs.
3. The EN system is closely related to "flow models". As a consequence, linear algebraic techniques can be applied for the quantitative analysis of the model.
The simple formalism introduced so far allows us to explore the basic relations between events in a contact-free net, namely sequence, conflict, concurrency and confusion.

(a) *Sequence Event:* $e_1$ can occur at $c$, but not $e_2$. However, after event $e_1$ has occurred then $e_2$ can occur (Figure 2).

(b) *Concurrency:* If in case $c$, two events $e_1$ and $e_2$ have concession in $c$ and $\#e_1 \cap \#e_2 = \phi$, then $e_1$ and $e_2$ can occur concurrently in $c$. Hence, in general, a case is transformed into a new case by a partially ordered set of occurrences of events (Figure 3).

(c) *Conflict.* Two events are in conflict in a case if both have concession, but have at least one preset place in common. If the two events are in conflict in a case, then in that situation, either one of them may occur, but not both (Figure 4).
(d) *Confusion*: In a case c, the events $e_1$ and $e_2$ can occur concurrently. However, with the occurrence of $e_2$, the transition $e_1$ can get into or out of conflict with yet another event $e_3$.

This situation of confusion is depicted in Figure 5.

A behavior of a net consists of all the processes it can give rise to. A process is a partially ordered set of events with attendant holding of cases.
II.B Place Transition Systems (P/T Systems)

P/T systems have been traditionally called Petri Nets. In the following definitions, we underline the static and the dynamic aspects of P/T systems.

Definition 4. (P/T System) — A Static View

A sixtuple $\Sigma = (S, T; F, K, W, M_0)$ is called a place/transition system (P/T-system) iff:

4.1 $(S, T; F)$ is a net where the $S$-elements are called places and the $T$-elements are called transitions.

4.2 $K : S \rightarrow N^+ \cup \{\infty\}$ is a capacity function.

4.3 $W : F \rightarrow N^+$ is a weight function.

4.4 $M_0 : S \rightarrow N$ is an initial marking function which satisfies $M_0(s) \leq K(s)$ for all $s \in S$.

Definition 5. (Marking, Follower Marking, Firing of a Transition in a P/T System) — A Dynamic View of P/T Systems

Let $\Sigma = (S, T; F, K, W, M_0)$ be a P/T-system.

5.1 A function $M : S \rightarrow N$ is called a marking of $\Sigma$ iff $M(s) \leq K(s)$ for all $s \in S$, where initial functions is $M_0$.

5.2 A transition $t \in T$ is enabled at $M$ (or has concession at $M$) iff $\forall s \in *t W(s,t) \leq M(s)$ and $\forall s \in t^* M(s) \leq K(s) - W(t,s)$.

5.3 If $t \in T$ is a transition which is enabled at a marking $M$, then $t$ may occur, yielding a new marking $M'$ given by the equation: $M'(s) = M(s) - W(s,t) + W(t,s)$ for all $s \in S$.

5.4 The occurrence of $t$ changes the marking $M$ into the new marking $M'$; we may denote this fact by $M(t)M'$ or by $M \xrightarrow{t} M'$. 
Remarks

1. The basic concept of a P/T system is that the S-elements which are now called places may carry any number of identical tokens. The state of the system is now called a marking and it is determined by the distribution of tokens in the places.

2. Nets and Net Systems are dual concepts in our terminology. A Net System is a Net with an initial marking. Consequently a P/T Net is a quintuple \( N = (S, T; F, K, W) \) (see definition 4).

3. We focus our attention on a special class of P/T systems, namely those with an infinite capacity function \( K(s) = \infty \) \( \forall s \in S \).

4. P/T systems can be viewed in two different ways:
   - As generalization of EN systems. This is the conventional point of view. When this viewpoint is adopted, the rich body of results formulated for EN systems cannot be applied to P/T systems.
   - As a short-hand notation for EN systems. If we take this approach, every EN system can be condensed into a P/T system and every P/T system is a concise representation of an EN system.

We can extend the above concepts to define the set of follower markings for a given marked P/T net. We define the Reachability set \( R(N, M) \) of a marked net \( N \) with a marking \( M \) to be the set of all markings which are reachable from \( M \). The reachability relationship is the reflexive transitive closure of the immediate follower relationship.

Definition 6. (Reachability Set)

The reachability set \( R(N, M) \) for a net \( N = (S, T; F) \) with marking \( M \) is the smallest set of markings defined by
6.1 $M \in R(N,M)$

6.2 If $M' \in R(N,M)$ and $M' > M''$ then $M'' \in R(N,M)$

Definition 7. (Reachability Tree)

The reachability tree represents the reachability set of a net. The nodes in such a tree are the various possible markings and the arcs are the transitions that connect the markings.

For a net with unbounded capacity, the reachability set could be infinite. Beginning with an initial marking the number of tokens in a place could go on increasing indefinitely if the place participates in a sequence of transitions, which is in the form of a loop. Two means are used to keep the reachability tree to a compact and finite representation. We represent the infinite markings that result from loops by using a symbol $\omega$, which can be though of as "infinity" and which represents a number that can be made arbitrarily large. The other way is to notice "duplicate" nodes in a reachability tree and represent all duplicate nodes with the same marking by a single node. It can be proved that the reachability tree of a net is finite. Readers may consult [40] for a formal proof. It should be noted that undecidability problems associated with a Petri net stem from the use of $\omega$ symbol, which hides information as to the exact integer sequence of tokens that are possible in a place. In our modeling of AI systems, loops will be treated as an error in representation. We will discuss methods to determine the presence of loops and their removal from a net representation in a subsequent section. Absence of loop removes problems associated with undecidability issues in a net.

Definition 8. (Incidence Matrix)

Let $N = (S,T; F,K,W)$ be a pure P/T net. For the sets $S$ and $T$ an arbitrary but fixed order is assumed:

$$S: s_1 < s_2 \cdots < s_m$$
\[ T: t_1 < t_2 \cdots < t_n \]

where \( m = |S| \) and \( n = |T| \).

8.1 A column vector \( v: S \rightarrow N^+ \) indexed by \( S \) is called \( S \)-vector of \( N \).

8.2 A column vector \( w: T \rightarrow N^+ \) indexed by \( T \) is called \( T \)-vector of \( N \).

8.3 A matrix \( N: S \times T \rightarrow N^+ \) indexed by \( S \) and \( T \) such that

\[
N(s_i, t_j) = W(t_j, s_i) - W(s_i, t_j)
\]

is called the incidence matrix of \( N \).

Obviously every transformation of a marking \( M \) into a follower marking \( M' \) can be linearly represented as

\[
M + N \ast W = M'
\]

where \( N \) is the incidence matrix and \( W \) is a \( T \)-vector. The converse, however, is not true.

We stress the dual concepts of structure and behavior of P/T net systems. The structure relates to marking independent properties which are determined solely by the underlying net. These properties are determined only by the flow relations which describe how places and transitions are interconnected.

The behavior of P/T net systems is by the marking dependent properties of the system related to the token flows. The concepts relevant to the behavior of a net system are: the reachability set describing the set of reachable markings, the reachability graph, the reachability tree, the synchronic distance between transitions, etc. The analysis of the behavior of a marked net is generally a much harder problem than the analysis of its structure.

We outline now one of the methods used for the analysis of P/T nets, the method of
invariants. Clearly the net structure enforces a certain behavior of a P/T system. Two aspects of this behavior are of primary concern, namely whether tokens can be lost in an uncontrolled way and whether it is possible to reproduce markings. Both properties can be investigated using linear algebraic techniques.

Definition 9. (S and T Invariants)

Let \( N = (S, T; F, K, W) \) be a P/T net and \( I \) be an S-vector of its incidence matrix \( N \) and \( J \) be a T-vector.

9.1 \( I \) is called an S-invariant of \( N \) iff \( I^T \cdot N = 0^T \).

9.2 \( J \) is called a T-invariant of \( N \) iff \( N \cdot J = 0 \).

9.2 \( P_I \subset S \) such that \( P_I = \{ s \in S | I(s) \neq 0 \} \) is called the support of \( I \).

9.3 \( T_I \subset T \) such that \( T_I = \{ t \in T | J(t) \neq 0 \} \) is called the support of \( J \).

9.3 A subnet \( N_I = (S_I, T_I, ; F_I, K_I, W_I) \) is called the graphical representation of \( I \) iff \( S_I \) is the support of \( I \).

\[
T_I: = *S_I \cup S_I^*
\]
\[
F_I = F \cap ([S_I \times T_I] \cup (S_I \times T_I))
\]
\[
K_{I(s)} = K(s), \quad \forall s \in S_I
\]
\[
W_{I(f)} = W(f), \quad \forall f \in F_I.
\]

The graphical representation with respect to support \( J \) is similarly defined.

Remarks

1. Every integer linear combination of S-invariants is an S-invariant. Every integer linear combination of T-invariants is a T-invariant.
2. It can be proved that the inner product of an $S$-invariant and the follower marking of some initial marking is an invariant quantity. If $M' \in R(N,M)$ then $I^T \cdot M' = I^T M$. It follows that the graphical representation of a non-negative $S$-invariant is a subnet in which no token can be lost or gained in an uncontrolled way, since $I^T \cdot M'$ is an invariant quantity for all follower markings of the initial marking $M$.

3. If $M$ is a reproducible marking of a P/T net, then the transitions occurring during the reproduction of $M$ are the transitions of the graphical representation of a $T$-invariant.

Conclusion -- The graphical representations of $S$- and $T$-invariants are subnets of the original net with a predictable behavior.

II.C High level nets (HL nets)

High Level nets have been developed to allow the net theory to represent formally entities with changing properties and relations. The tokens in HL nets have attached to them records of attributes as opposed to the tokens in P/L nets, which are indistinguishable. HL nets allow manageable representations of complex systems. They allow folding of identical subnets of P/T nets into a single subnet and of the same time, preserving the possibility to distinguish between different processes.

The first family of HL nets are the Predicate/Transition nets (PrT nets) introduced by Geinrich and Lautenbach [50]. In PrT nets, information can be attached to each token as a token-color and each transition can occur in different occurrence-colors. When a transition occurs, the relation between the occurrence-color and the involved token-colors are defined by expressions attached to arcs. Restrictions upon the possible occurrence-colors can be defined as predicates attached to a transition.
In the following, we give the definitions for a family of HL nets, the Colored Petri nets, (CP-nets) introduced by Jensen [23]. There are two equivalent ways to introduce CP-nets, by defining the incidence matrix, the CP-matrix or by defining the CP-graph. Algorithms to translate from one form into another are given in [23]. We define only CP-matrices here.

Definition 10. CP-matrix.

A CP-matrix is a 6-tuple \( N = (S,T,C,I_-,I_+,M_0) \) where

10.1 \( S \) is a set of places.

10.2 \( T \) is a set of transitions.

10.3 \( S \cap T = \emptyset \) and \( S \cup T \neq \emptyset \).

10.4 \( C \) is a color-function defined from \( P \cup T \) into a collection of non-empty sets representing colors. It attaches to each place a set of possible token-colors and to each transition a set of possible occurrence-colors.

10.5 \( I_- \) and \( I_+ \) are the negative and the positive incidence-functions defined such that

\[
I_-(p,t) \quad \text{and} \quad I_+(p,t) \in \{ C(p)_\text{lin} \rightarrow C(p)_\text{lin} \}_L
\]

for all \( (p,t) \in S \times T \). (Subscript \( L \) stands for linear transformation.)

10.6 \( \forall p \in S \) there is a \( t \in T \) such that \( I_-(p,t) \neq 0 \) or \( I_+(p,t) \neq 0 \) and \( \forall t \in T \) there is a \( p \in S \) such that \( I_-(p,t) \neq 0 \) or \( I_+(p,t) \neq 0 \).

Note: this requirement relates to standard definition of Petri nets, which do not allow isolated places of transition.

10.7 \( M_0 \) is the initial marking and it is a function defined on \( S \) such that
\[ M_{0}(P) \in C \in P_{MS} \text{ for all } p \in S. \]

Remarks

1. A multi-set over a non-empty set \( C \) is a function \( C \rightarrow N \) where \( N \) is the set of all non-negative integers. A multiset can contain multiple occurrences of the same element. The set of all finite multisets over the non-empty set \( C \) is denoted by \( C_{MS} \).

2. The problem of finding \( S \)-invariant for a CP matrix is slightly more difficult than the case of P/T nets, since the matrix elements are not contained in a field, hence there is no general algorithm to solve homogeneous equations. A brute force method could unfold the CP-net into a large P/T net, find its invariants (usually a large number), then fold them into CP-invariants. No algorithm is known for the last step. Another solution [23] is to apply a set of transitions, which exploit the properties common to CP-matrices, which are very sparse, non-square, have high degree of dependency between individual columns, and have many matrix elements which are either identity functions or simple communication functions. A transformation must be sound, i.e., it should not change the set of \( S \)-invariants.

3. Construction of reachability trees for CP-nets is equally challenging. The basic property of CP nets is that they possess, very often, classes of equivalent markings. In such a case for each equivalence class, only the subtree of node is developed.

II.D Concluding remarks

Several families of net models have been described, ranging from Elementary Nets to High Level Nets. We adopt the point of view that higher level nets are simply a more convenient way to represent systems, since they lead to more concise representation. A higher level net model can always be unfolded into a more primitive one. The analysis of
higher level nets is more intricate, as we have seen when discussing reachability trees and invariants for CP nets. It is important that when new nets are defined, in order to represent more conveniently a certain type of system, the firing rules not be modified. When the firing rules are modified, the incidence matrix is no longer able to capture the net structure.

There are several methods for the investigation of nets, namely the study of the reachability set, transformations by isomorphisms, and the method of invariants. In the latter case, analysis of a net can be performed on subnets ignoring the behavior of the whole system. This is a general method which can be applied from the simplest nets to the high level nets. This method seems to be very promising for the analysis of systems with prohibitively large state space. Net theory allows us to relate behavioral properties to structural properties like connectedness. It is the goal of the net theory to study the behavior of a net, as it results from the net structure.

The net models discussed so far do not reflect the concept of time. It is assumed that transitions fire instantaneously. Most of the properties of the systems like absence of deadlocks, can be investigated in this framework. Timed nets are nets in which a deterministic, or a random time elapses, from the instant when a transition is enabled, until the time it fires. Stochastic Petri nets used for performance modeling and analysis belong to the class of timed nets. Since we intend to use a net model for mapping to a parallel system, a timed net will be used.

Mutual dependency and independency of transitions in a net is an important issue for design and analysis of any system. The concept of synchronic distance provides us with a metric that serves to quantify the relationship of two transitions in terms of dependency. Several definitions of the synchronic distance can be found in the net literature. A common approach [41] is to define synchronic distance between two transitions $t_1$ and $t_2$ as a
measure of how often the two transitions do not occur together in different processes. (A process is a path in a reachability tree). The absolute difference of their relative occurrence frequency in a process \( p \) is represented as \( \text{Var}(t_1, t_2, p) \). The supremum of these \( \text{Var} \) values in all processes is called the synchronic distance between \( t_1 \) and \( t_2 \) and it provides a quantitative measure of degree of dependence between the two transitions.

The net as a computational model has been shown to be equivalent to the Turing machine \([45]\). It has been used to model solutions to problems like hamiltonian circuit, calculation of recursive functions, shortest path problems in a directed graph \([49]\), etc. The net representation therefore, provides a uniform semantics to a wide class of computational problems such as numeric functions, logical systems, and models of dynamic and distributed systems. Our approach to using a net representation of an AI system for mapping it onto a given multi-processor architecture, is equally applicable to net representation of any hybrid system of computation involving both symbolic and numerical computation. This, in our view, constitutes a major strength of using net formalisms for describing computational models.

Finally, it should be noted that nets allow the possibility of analyzing the system at different levels of abstraction. A transition at a given level can be expanded to a subnet at a lower level. In Section V, we use the net representation to show interaction between rules and also show the interaction between terms within a single rule. The same representation scheme will show concurrency in operations at the level of rules, as well of concurrency of operation within a single rule.

**III. HORN CLAUSES**

We begin with net representation of propositional logic without negated clauses. In particular, we provide a net model for describing a system of horn clauses. A *Horn Clause* is an ‘implication’ type of clause with one or more antecedent propositions and
only one proposition in the conclusion. In this section we limit ourselves to horn clauses with positive propositions; in the next section we discuss systems of horn clauses having both negative and positive propositions.

We represent a proposition as a place in the network. Propositions are linked together as horn clauses by transitions and arcs between them, with a transition representing an implication clause. The direction of the arcs specify the direction of implications and is the direction in which inference is propagated in the network. To model systems of propositional logic we need to introduce a suitable local property to some elements of the net viz. places which we have termed activation value. At any point of time a place is characterized by two attributes: marking and activation value. Firing of a transition leads to changes in the markings of its preset and postset. However, in the case of activation values only the values of the postset places are affected. A transition or change function for activation values, like the markings of a system, is formally provided in the system definition.

The activation values in the net correspond to the truth values in a propositional logic system. Formally, a net with the above characteristics can be defined as below:

Definition 10. (Colored Logical Net).

A eight-tuple \( CLN = (S, T; F, K, W, M_0, A, \tau) \) is called colored logical net if and only if:

10.1 \( (S, T; F, K, W) \) is a P/T net (as defined in Section II) with \( M_0 \) the initial marking.

10.2 \( A : S \rightarrow \{0, 1\} \), is called the activation function.

10.3 \( \tau : T \rightarrow R^+ \), is the time function. It specifies the time taken by a transition to fire once it has been enabled. In this section, we assume that all transitions
10.4 The new activation value of a place \( s \), as a result of firing a transition \( t \), is a function of: (i) its old activation value, (ii) and the minimum of the activation values of the preset places of \( t \). If the activation function \( A \) changes to \( A' \) as a result of firing of a transition \( t \), \( A' \) is defined as follows:

\[
A'(p) = \max \{A(p), \min \{A(s) \mid s \in \#t\} \} \quad \text{for all} \quad p \in \#t, \quad \text{otherwise}
\]

\[
A'(p) = A(p).
\]

As a result of the above definition, the activation value of a place cannot decrease with time. Presence of an activation token at a place implies an activation value of 1, otherwise the activation value is 0.

10.5 At any case \( c \subset S \), a place \( p \in c \) has two attributes: \( M(p) \) and \( A(p) \). This is implemented by having tokens of two colors, marking tokens, and activation tokens.

10.6 All transitions in CLN are of a single color (i.e., single type).

**Definition 11:**

Let CLN be a colored logical net, then

11.1 A place \( p \in S \) with no incident arc is called a *root* node.

11.2 A place \( p \in S \) with no outward arc, is called a *goal* node. A subset of goal nodes which are of particular interest will be called *preferred* nodes.

A hom clause such as \( A \& B \rightarrow C \), will appear in a colored logical net representation as follows. We have a place for each proposition and they are marked as such. If the propositions \( A \) and \( B \) are 'true' the initial activation values are set equal to 1 and to begin with there is a marking token at places \( A \) and \( B \), i.e., the initial marking is \((1,1,0)\). The initial activation values are \((1,1,0)\), i.e., there is an activation token at places \( A \) and \( B \).
When transition $t$ fires, the revised activation value of place $C$ is 1, i.e., a new activation token is placed at $C$. The follower marking of marking tokens is $(0,0,1)$ and of activation tokens is $(1,1,1)$. Since the activation values are now $(1,1,1)$ all the propositions are 'true'. It may be noted that the transition $t$ acts as an 'AND' node. It can fire if and only if all the preset places of the transitions $t$ are occupied. We can also view the resultant marking of place $C$ in the clause $A \& B \rightarrow C$ as follows. Let the markings of place $A$ and $B$ be $(\lambda_a, \alpha_a)$ and $(\lambda_b, \alpha_b)$ then the markings of $C$ is $(\lambda_a \wedge \lambda_b, \alpha_a \wedge \alpha_b)$.

A. Forward Inference

Forward inference is known as modus-ponens in logic. It is also known as bottom-up inference [27] and is similar to forward chaining in expert systems. In forward inferencing, the process is propagated from the conclusion of a horn clause to the antecedent of another horn clause with matching terms.

We now describe an algorithm to do forward inferencing in parallel on a system of horn clauses represented as a colored logical net. (An example of the algorithm is shown in Figure 7). It is assumed that there are enough processors to allocate one horn clause per processor. The places of a net are implemented in the local memory of the processors. The nature of communication in the network is assumed to permit processors to access the local memory of its neighboring processors. Since a given place can be part of the preset or postset of several transitions (processors), processors communicate by sharing places as shared variables in a common memory space.

In Section VII, we show how the processing can be done in parallel with fewer processors without sacrificing concurrency that is available in the system. We also describe an algorithm for allocation of clauses to processors based on analyses of the net representation of the system.
ALGORITHM 1:

Step 1: Initialization:

(1) Represent each fact 'D' as a root node of the net

\[ D \]

(2) For each clause like this \( A \land B \land \ldots \) (a finite set of propositions) \( \rightarrow C \), draw a place transition net like this:

\[ A \quad B \rightarrow C \]

(3) Draw an arc/transition/arc to connect places which are matching, i.e., a place which is conclusion in one clause and an antecedent of another, or is a 'fact' node. The direction of arcs is from the conclusion side to the antecedent side. Mark the goal nodes and the root nodes. Identify the places and the transitions by two different sets of ordered indices.

(4) Set the activation values of all propositions which are 'true' to 1 by placing an activation token in the corresponding places. Set marking tokens equal to the number of outgoing arcs at each of the 'true' proposition-places.

(5) Set the weight of each \((p,t)\) arc as 1. Set the weight of each \((t,p)\) arc as equal to the number of outgoing arcs from place \( p \). This will ensure that there are always as many tokens as are transitions that one can visit from place \( b \). Therefore, there is no scope of conflict in this net.
Step 2: While time < TIME let all transitions DO the following in parallel:

(i) Let all transitions that are enabled occur. Derive the revised markings and the activation values in terms of rules of a colored logical net.

(ii) Let all transitions check if they are enabled. If true, then go back to step (i).

ENDWHILE.

B. Properties of Algorithm

The activation values and the markings diffuse through the network in a single pass, that is, there is no backtracking in this system. Since, in our system, transitions take unit time, the total time required for completing the diffusion is proportional to the diameter of the net. (The diameter of the net is $O(|S| + |I|)$). The presence of a loop in the net is indicated by transitions taking place beyond this time limit, which is set equal to value TIME in the above algorithm. Loops indicate the presence of tautologies, i.e., goals which recur as their own subgoals. Since tautologies do not contribute to the solutions of problems, they can be deleted without affecting the consistency of a net [26].

Theorem 1: All propositions which can be proved to be 'true' in first-order propositional logic are proved true by Algorithm 1. Algorithm 1 terminates.

Sketch of proof: A transition fires when its preset places have sufficient marking tokens. A transition sets its conclusion to be 'true' (i.e., its activation value is 1) only when all its preset places (antecedent propositions) have activation value of 1, i.e., (are 'true'). Therefore it obeys the modus-ponens rule.

As there is no conflict in the net, all transitions that can fire, will eventually fire setting the postset places to be 'true', only when the preset places are all true. Since the activation values can never be reduced, the network eventually ends with the maximum
sum of activation values logically consistent with the initial conditions.

By the condition of the 'while' loop, Algorithm 1 obviously terminates.

A formal proof would be based on induction and on the fact that our system propagates inference strictly according to the rule of modus-ponens in first-order logic.

C. Backward Inference

Backward inference is known as modus-tollens in logic. It is also known as top-down inference [27] and is similar to backward chaining in expert systems. In backward inferencing, an inference is propagated forward from terms in the antecedent of a horn clause to matching terms in the conclusion of another clause.

The net representation introduced above for a system of horn clauses corresponds to forward inference. A backward graph can be generated as well by having all arcs in the reverse direction. This corresponds to generating follower markings where the incidence matrix $N$ is changed to $-1 \cdot N$. With this new representation we can apply Algorithm 1 to do backward inference on the system. As far as initial markings and activation values are concerned we set the goal propositions whose proofs are desired as marked with activation and marking tokens.

An example of Algorithm 1: (refer to Figure 7).

A set of fact and rules:

\[
\begin{align*}
A, B \\
B \rightarrow C \\
A \rightarrow D \\
C & \& D \rightarrow E
\end{align*}
\]
$C \rightarrow F$

- a place with no marking or activation token

- a place with both marking and activation token

- a place with only activation token

**INITIALIZATION (Step 1)**

**After firing of all transitions (Step 2-4)**

Figure 7.

IV. NON-HORN CLAUSES

A. Clauses With Negative Propositions
In this section, we describe representation of general implication type clauses without restrictions. At first we generalize horn clauses to allow for negative propositions. We use the mechanism of the colored logical net as described earlier. In this net we have two types of activation tokens, i.e., two colors for such tokens. We call them colors $f$ and $g$. A positive proposition will be represented by a place of that name, but will have incident arcs and outgoing arcs that permit only activation token of color $g$ to flow along these arcs. Similarly a negative proposition will correspond to a place which will have arcs that allow only activation token color of $f$ to flow along those arcs.

A proposition which has both positive and negative instances in the system is represented as two different sets of places which allow different colors of activation tokens. A situation of contradiction will be shown by places corresponding to the same proposition having activation tokens of both colors, i.e., $g$ and $f$. In this system we can show three types of 'truth' conditions, 'true' corresponds to presence of activation tokens of color $g$, 'false' corresponds to tokens of color $f$, and absence of activation tokens shows indeterminancy of the truth value for that proposition in the system.

A simple instance of such a system is shown here below (refer to Figure 8).

\[ A \land B \rightarrow \neg C \]
\[ \neg C \rightarrow D \]
\[ C \land G \rightarrow H \]
\[ E \rightarrow C \]
\[ C \rightarrow F \]
\[ F \rightarrow G \]
C and C' in Figure 8 correspond to the negative proposition and therefore permit only activation tokens of color f to arrive at that place. (Note that negative and positive places corresponding to the same proposition are not connected to each other even if they are matching terms across clauses, because they are treated as two different propositions.) The positive proposition C is represented by places C', C''' and C''''.

Case 1: If propositions A and B are true, then D will be true and H and F will be indeterminate.

Case 2: If propositions A, B, C are true, then C will be both false and true because place C will have a token of color f and place C'' will have a token of color g, showing contradiction.

Figure 8.

Definition: (Modified Colored Logical Net)

A five-tuple MCLN = (S,T;F,K,W,M,A,τ) is called Modified colored logical net if and only if:
12.1 $(S,T;F,K,W)$ is a P/T net with $M_0$ as initial marking.

12.2 $A: S \rightarrow \{0,1\}$ is called the _activation function_.

12.3 The new activation value of a place $s$, as a result of firing a transition $t$, is a function of (i) its old activation value, (ii) and the minimum of the activation value of the preset places of $t$. If the activation function $A$ changes to $A'$ as a result of firing of a transition $t$, $A'$ is defined as follows: $A'(p) = \max (A(p), \min \{A(s) | s \in \star t\})$ for all $p \in s$; otherwise $A'(p) = A(p)$. Presence of an activation token at a place signifies an activation value of 1, otherwise the activation value is 0.

12.4 At any case $c$, a place $p \in c$ has two attributes: $M(p)$ and $A(p)$. This is implemented by having tokens of three colors, marking tokens of color $m$, activation token of color $g$ and activation token of color $f$. The disposition of marking token as a result of a transition is given by the rules of a P/T net. The number of activation tokens generated at a place is given by (12.3) above. The color of the activation tokens generated is given by the functions inscribed on the arcs. The functions are simple and specify token color that can flow along the arcs. A negative proposition allows only color $f$ activation tokens to flow in and out of the corresponding place. Likewise a positive proposition allows only activation tokens of color $g$ to flow in and out of that place. Therefore no place can have activation tokens of both color $f$ and $g$.

12.5 All transitions in MCLN are of single color (i.e., of a single type).

12.6 $\tau: T \rightarrow R^+$, is the _time_ function. It specifies the time taken by a transition to fire once it has been enabled. In this section, we assume that all transitions take a unit time.
Algorithm 1 can be modified to allow for processing on a MCLN. In Section VII, we will discuss how this algorithm can be implemented with fewer processors, while still exploiting the possible concurrency in the system.

**Theorem 2:** Algorithm 1 implemented on the basis of a MCLN will find a contradiction in a set of clauses if and only if a contradiction is implied by first order propositional logic.

*Sketch of proof:* The proof is based on the fact that there is no conflict in the net. Every transition that has its 'antecedent' true will have marking and activation tokens in the antecedent places and therefore it can fire. When such a transition fires it will make its conclusion place have an activation token implying that it is true. Therefore all places that can receive activation tokens as per the rules of modus-ponens will receive activation tokens in the system. This includes tokens of different colors as well. Hence if a proposition has both negative and positive instances in the system and if it can be proved both false and true in a first-order propositional logic, its corresponding places in the net will receive tokens of both the color $g$ and $f$ thereby proving contradiction.

**B. Clauses With Multiple Consequent Propositions**

Clauses with multiple consequent can be of the following three generic types:

(i) $A \land B \rightarrow D$ or $C$ (inclusive 'or'),

(ii) $A \land B \rightarrow C \land D$,

(iii) $A \land B \rightarrow D$ or $C$ (exclusive 'or', also called disjunction type of clauses).

For ease of explanation, we consider one generic case at a time. For the first two types, the clause can be represented in our system by two separate clauses: (i) $A \land B \rightarrow C$ and
(ii) $A \& B \rightarrow D$. The above clauses can be represented as a system of places and transitions as discussed earlier.

For the third type, i.e., disjunctive clauses, it can be represented as a system of places and transition as follows (refer to Figure 9).

![Figure 9](image)

The dummy place $X$ is called a conflict node and transitions $t_1$ and $t_2$ are conflict transitions. $X$ is now an 'OR' type of node and only one transition, $t_1$ or $t_2$ can fire. To propagate inference over the net, we need either a mechanism for resolving the conflict or a mechanism to search the space of all possible firing sequences. In the first case we need to develop the tree structure that depicts the sequences of transitions possible and then search the tree in a parallel fashion by some parallel version of the A* algorithm [39]. In this paper we will use the mechanism of the colored net to implement implication clauses with disjunction.

Let us consider a net representation of such an exclusive 'OR' type of non-horn clause system as shown in Figure 10, with a system of facts and disjunctive clauses as follows:

$P, M$
There are four conflict nodes A, B, C and D. For each conflict node, only one out of two conflict transitions can fire. We construct our system conflict-free as before by suitable weighting of the arcs. We have as many colors associated with tokens as there are possible sequences of conflict transitions. This color scheme can also be viewed as if the
tokens are carrying an attribute list, the elements of the lists being the conflict transitions it has participated in.

Every marking token that is generated at a conflict place includes in its attribute list the identification of the transition that generated it. When this token participates in other transitions, it passes this identification to all tokens generated in those transitions. Each token, therefore, carries the history of the sequences of conflict transitions that led to its generation. A marking token arriving at place Q will have a color corresponding to the conflict transition A1. A marking token arriving at place K will have a color which corresponds to the conflict transition sequence, A2, B1, C2. In the above example, all the goal nodes T, H, I, K and L will have marking tokens, each having a color that specifies the choice of transitions at each disjunctive clause.

Algorithm 1 can be applied to this system of colored net. All places that have activation tokens are 'true', subject to the choices of transitions which is provided by the color of the marking token. The marking color specifies the choice of transitions that can make the proposition true and this sequence can be checked to see if it makes sufficient semantic sense. This mechanism of carrying out inference over disjunctive clauses is, however, not without a price. Checking the color of tokens to see if they have resulted from a sequence of semantically non-contradicting choice of transitions involves a combinatorial search that can be computationally expensive.

C. Belief Networks

The CLN mechanism can be easily amended for representation of non-binary logic systems. In non-binary logic, a predicate is associated with a truth value which may be other than 0 or 1. This can be done by having activation tokens of various colors, each color corresponding to a real number in the interval (0,1), i.e., we have an 'uncountable' number of colors for these activation tokens in such systems.
Various ad-hoc rules and functions have been suggested in the expert system literature for deriving the truth value of a consequent from the truth values of the terms in the antecedent. Readers may consult [21] for a review of such rules. These rules or functions can be incorporated in our system by suitably modifying the change-function of activation tokens as related to transitions firing.

D. Conclusion

Non-Horn clauses, i.e., clauses with disjunction or multiple consequents constitute a natural representation of causal relationships. During the formulation of an expert system, it is common to come across relationships that have embedded in them other relationships. Relationships that involve other relationships is a common way in which we describe the world. Clause systems like Prolog or expert systems, by disallowing such representation, have their power of expression severely limited.

An instance of such relationship is: if the market is quality conscious (M), then the relationship of increased quality effort (Q) and increased sales (S) holds true.

This is an implication form as follows: M → (Q → S) or M → Q or S. (This ‘or’ is a disjunctive type).

If M is true, we will expect either (i) S to be true and ~Q to be false, i.e., both Q and S is true or (ii) ~Q to be true and S to be false, i.e., both S and Q to be false.

Clauses with disjunctive form also permit us to specify contextual knowledge. An instance of such a clause is – if the annual salary is 40K then the employee is either a junior manager or a senior supervisor. In Horn clause system or in a typical expert system, a disjunctive clause A → B or C, will have to be represented as follows:

\[ A \& (\text{some factors}) \rightarrow C \]

and
These factors, when they can be discussed, will have to be differentiated, which usually involves numerous combinations of factors applicable to the overall context. We need to define rules to make choices depending on the various combinations of missing data and contextual facts, which can lead to exponential numbers of rules.

The difficulty of anticipating and specifying a complete and accurate set of differentiating factors, is called the 'frame' problem in AI. It is the principal cause of 'fragility' in production systems. Horn-clause techniques in Prolog and expert systems avoid the combinatorial search that is necessary in our system by shifting the burden to the knowledge engineer in terms of exponentially increasing the set of rules and variables, which ultimately slows down the performance of the system.

Thus we see that the ability to represent rules with multiple consequences allows us a more concise and complete specification of the system, but only at the expense of greater search during the problem solving phase. There is obviously a great need for more research on use of disjunctive clauses in production systems. Default reasoning [15] and truth-maintenance systems [10], address some of these concerns.

It is interesting to note how Prolog type inference systems differ from net based systems, as described here, in their ability to deal with negation. Net systems handle negation by treating negative and positive instances of a proposition as entirely separate entities, which are not related to one another during the inference phase. Prolog systems handle negation by having a unary operator not defined as a primitive operator in the language system. Prolog being based on the closed-world assumption [6] views anything not provable as 'false'. In the Prolog system not is not exactly equivalent to 'false' in first-order logic, it only implies failure to satisfy a proposition or a clause. The cut-fail combination achieves a semantics closer to falsehood, but only at the major price of
rendering a declarative language to a procedural one, because the position of the cut-fail clause is all important in fixing the right semantics in the system. The net based systems perform better by having three different truth values, true, false and indeterminate, thereby keeping falsehood and indeterminate as two classes apart.

V. EXPERT SYSTEMS

A. Clause and Rule-Based Systems

Our interest in this section is the demonstration of the use of relation net mechanism for implementing an inference mechanism in a Prolog type system. We can also implement systems of predicate calculus or expert systems by the same mechanism of the relation net. In a relation net the tokens are structured objects like records having fields and values. The fields have certain 'types' and allow values from a restricted domain set. Each place is associated with only one type of token (i.e., a single record type). The arcs are labelled with tuples of variables. These tuples are related to the tokens residing in the places that these arcs connect to. Transitions are inscribed with logical relations which relate the tuples inscribed on the arcs. Formal definition of a relation net is as follows:

Definition 13: (Relation Net)

A four-tuple \( RN = (S,T;F,K,W,\Sigma,\pi) \) is called a relation net if and only if

13.1 \( (S,T;F,K,W) \) is a place/transition net.

13.2 A structure \( \Sigma \), defines a collection of typed objects together with some operations and relations applicable to them. Formulas built up in \( \Sigma \) can be used as inscriptions on the transitions.

13.3 A labelling of an arc, which is connecting between a place and transition, with
the formal sums of tuples of variables. The length of each such tuple is the
arity of the tokens in the place which is connected to this transition by this
given arc.

13.4 $\tau: T \rightarrow \mathbb{R}^+$ is the time function. We assume that the time taken by a transition
is related to the number and arity of tokens that participate in a transition. This
will reflect the time taken to perform matching among terms for common vari­
ables and constants.

A token $t = (a, b, c)$ in a place $P$ denotes the fact that the relation $P(a, b, c)$
corresponding to that place is true for that particular instantiation of the tuple of arguments
contained in that token.

In order to demonstrate how the proposed net mechanism can carry out deduction in
an expert system, we consider an example of such a system from Nilsson [33]. This par­
ticular example has been discussed in [16], where the use of $S$-invariants for analyzing this
system of rules and facts is also described. We express the rules in a language similar to
OPS-5, a popular production system for building expert systems.

An example of expert system implementation (refer to Figure 11).

(i) Rule 1: $\text{manager}(^\text{man}: x ^\text{dep}: y) \rightarrow \text{works in}(^\text{worker}: x ^\text{dep}: y)$

(ii) Rule 2: $\text{works in}(^\text{worker}: y ^\text{dep}: v) \& \text{manager}(^\text{man}: x ^\text{dep}: v)$

$\rightarrow \text{boss}(^\text{man}: x ^\text{worker}: y)$

(iii) Rule 3: $\text{works in}(^\text{worker}: z ^\text{dep}: v) \& \text{works in}(^\text{worker}: y ^\text{dep}: v)$

$\rightarrow \text{not married}(^\text{worker}: y ^\text{worker}: v)$
(iv) Rule 4: \(\text{married}(\text{worker:y} \ \text{worker:z}) \rightarrow \text{married}(\text{worker:z} \ \text{worker:y})\)

(v) Rule 5: \(\text{married}(\text{worker:y} \ \text{worker:z}) \& \text{worksin}(\text{worker:y} \ \text{dep:PD})\)
\[\rightarrow \text{insuredby}(\text{worker:y} \ \text{company:EC})\]

(vi) Working memory elements at the beginning are:

manager(\text{man:J} \ \text{dep:PD}), \text{manager}(\text{man:H} \ \text{dep:SD}), \text{worksin}(\text{worker:T} \ \text{dep:PD}), \text{worksin}(\text{worker:S} \ \text{dep:PD}), \text{worksin}(\text{worker:P} \ \text{dep:PD}), \text{worksin}(\text{worker:M} \ \text{dep:SD}), \text{worksin}(\text{worker:B} \ \text{dep:SD}), \text{worksin}(\text{worker:J} \ \text{dep:SD})\) and \(\text{married}(\text{worker:J} \ \text{worker:M})\)

**INITIALIZATION**

![Diagram](image)

Figure 11.
As in MCLN, we represent each term of an implication clause by a place. The clause is represented by a transition and its associated places. The direction of the transition is the direction of implication. The inscriptions on the arcs specify the relationships to be satisfied by the token attributes for a transition to fire. For instance, transition $t_3$ can fire only when the two token tuples $[y,v]$ and $[z,v]$ from relation Worksin have their second attributes in common. To begin with, tokens are inserted in the net at the root nodes corresponding to the facts in the system. All enabled transitions are allowed to fire. All matchings that are possible are generated in the system (refer to Figure 12).

Answers to questions like: "Who is J's boss?" is obtained by looking at the goal place corresponding to the relation 'boss' and reading the first attribute value for the token that has the second attribute value equal to $J$.

Note here that our method of inferencing is similar to the method of resolution as adopted by the connection-graph proof procedure [26]. This procedure has also been suggested as a paradigm for conducting resolution in parallel [8]. In connection-graph procedure, implication clauses are written as directed graphs with the propositions as nodes. Matching terms, i.e., terms which are consequents in one clause and antecedents in another, are linked together by undirected links. After such a connection-graph is drawn, the resolvent procedure repeatedly selects a link, resolves upon that link by generating all associated resolvents and including the resolvents in the graph. A term that has been resolved, has all its links dropped and is deleted from the graph. The procedure continues till it can either not proceed any further or all clauses have been dropped.
The incidence matrix of this relation net, shown in Figure 13, helps us to generate the $S$-invariants of this system. The procedure is detailed in [14]. The $S$-invariants $p_1$, $p_2$ and $p_3$ are shown in Figure 14. Given initial marking $M_0$ shown in Figure 9, the following relation holds:

$$p_i \cdot M_0 = p_i \cdot M \forall M \in R(M_0)$$

(5)

Suppose we want to answer the question "Is J insured by EC?" and suppose we want to ascertain the minimum number of facts we must know to answer the question. The root places are A, B and C*; we are interested in a marking $M_0$ covering only these places. By
using $M_0$ as unknown and $M$ containing all zeros except $M(F) = [J, EC]$, and considering $S$-invariant $p_3$, one gets

$$[y, z][x, y][y, EC](M_0(A) + M_0(B)) = [y, z][x, y][y, PD][J, EC]$$

Treating this equation by methods described in [14], one obtains:

$$M_0(A) + M_0(B) = [J, PD] \text{ and } M_0(C) = [J, z]$$

The above equation has the interpretation that in order to assert the goal, we must know that either $J$ is manager, or $J$ works in PD. Also that $J$ should be married.

**INCIDENCE MATRIX**

<table>
<thead>
<tr>
<th></th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$-x, v$</td>
<td>0</td>
<td>$-x, v$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>$x, v$</td>
<td>0</td>
<td>$-y, v$</td>
<td>$-y, z$</td>
<td>$-y, PD$</td>
</tr>
<tr>
<td>C*</td>
<td>0</td>
<td>$-y, z$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>$[y, z] + [z, y]$</td>
<td>0</td>
<td>0</td>
<td>$-y, z$</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>$[x, y]$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$[y, z]$</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$[y, EC]$</td>
</tr>
</tbody>
</table>

**Figure 13.**
As stated earlier our interest here is to represent a collection of clauses that is implementable in Prolog. Parallel implementation of this system provides us with an entirely new framework for parallel implementation of Prolog-like systems. It should be noted that our system is in many ways fundamentally different from Prolog in that Prolog was conceived and implemented as a sequential and backward inference mechanism. Our system is of the forward inference type and is inherently parallel in nature. Flow control mechanisms like cut, etc., are not relevant here because our system is based on a complete matching approach. Some approaches on parallel implementation of Prolog such as that of "eager evaluation" described in [9], come close to our method.

The system of relations net can be adapted for representation of expert systems. Even though both expert systems and Prolog systems are based on implication type clauses, there are some fundamental differences between the two. Expert systems usually have multiple consequents in their clauses unlike Prolog. We have shown earlier how net based systems handle clauses with multiple consequents. Prolog, being a language based on first-order logic is also monotonic in character. This means that the list of facts and relationships proved to be true, can only grow monotonically with time. Whatever has
been proven true stays true for all time. Expert systems on the other hand are usually non-monotonic in nature. Whatever has been concluded now may stand withdrawn at a later time. This is both a cause of strength and weakness of such a system. This characteristic nature of an expert system is exhibited by deletion of working memory tokens in the system. Non-monotonic nature of expert systems is due to the presence of contradictory clauses, contradictory data. Note also that data is usually provided to the system in an interactive manner spread over time. Net based systems like expert systems only implement the if-then paradigm mechanically without any concern for any underlying semantic or theory and therefore can take contradictions in stride.

B. RETE: Matching Algorithm

Pattern matching is an ubiquitous feature in A.I. applications. For instance, it is a basic component of a production system. It is also the basic activity that has to be realized before a transition can take place in the relation net described earlier. Production system interpreters use it to determine which production rules have satisfied a condition, which is the same activity that a relation net has to perform to determine whether a transition is enabled or not. It has been estimated that pattern matching constitutes about 90% of the computation time in an expert system computation [34].

In A.I. literature, two algorithms for parallel execution of pattern matching have attracted a lot of attention. These are the RETE algorithm by Forgy [11] and the TREAT algorithm by Miranker [32]. In this paper, we limit ourselves to a net description of the RETE algorithm. In a production system interpreter, the output of the match process and the input to conflict resolution form a set called the conflict set. This is a collection of ordered pairs of the form

[Production, List of elements matched by its LHS].
The ordered pairs are called instantiations. The RETE match algorithm is an algorithm for computing the conflict set, for comparing the LHS of the production rules to a set of working memory elements in order to discover all the instantiations. Since most instantiations do not tend to change with additions or deletions of individual elements in the working memory set, it saves effort in pattern matching by carrying along in the memory the set of instantiations generated at any point in time and doing only incremental changes in the instantiation set with every change in the working memory set.

The instantiations of working memory elements are saved with each rule that it matches as a list. When a new memory element enters the working memory, all rules theoretically can pattern match in parallel with this new element and update the existing instantiation list. The updated instantiation list leads to updating of the conflict set in turn. The entire pattern matcher can be viewed as a black box with one input and one output (refer to Figure 15).

(Changes to working memory)  
\[ \text{BLACK BOX} \]  
(Changes to the conflict set)

Figure 15.

The description of working memory changes are passed on into the black box and are called tokens. A token is an ordered pair of a tag and a list of data elements. The tags can be + or - to signify if the change is an addition or deletion from the working memory list. Every change in the working memory list results in a copy of such a token being sent to all rules. The pattern matching of each rule proceeds in a tree like fashion. The root
node in the tree receives the new token. The successors of the top node, due to the intra-element tests, have one or more inputs and only one output. At each such node, copies of all tokens that have successfully instantiated is kept. Tokens that instantiate with other tokens flow down the tree. Some set of tokens which achieve complete matching of the rule, arrive in the last node. Presence of tokens in the last node signifies that the LHS of the rule has matched with the existing memory state. These rules are members of the conflict set. Every time a rule achieves this state, it is made a member of the conflict set and similarly with changes in memory element every time it fails to match, it is deleted from the conflict set. This way the RETE algorithm incrementally manages the conflict set.

For the rule and fact set considered earlier, a simple net representation of the matching algorithm is applied to the rule below.

\[
\text{WORKSIN}(y: \text{WORKER } v: \text{DEPARTMENT}) \& \text{MANAGER}(x: \text{MANAGER } v: \text{DEPARTMENT})
\]
\[
\& \text{DIVISION-DEPT}(v: \text{DEPARTMENT } d: \text{DIVISION})
\]
\[
\rightarrow \text{BOSS-DIV}(x: \text{WORKER } v: \text{MANAGER } d: \text{DIVISION})
\]
Working memory tokens relating to the relation WORKSIN is inserted into the place as shown above. Similar tokens are inserted into the place MANAGER and DIVISION-DEPT. Matching between tokens WORKSIN and MANAGER is done at transition \( t_1 \), the resultant matched tokens are located at place \( P_1 \). Tokens between place \( P_1 \) and memory tokens inserted into the place DIVISION-DEPT are matched by the transition \( t_2 \) and the resultant matched tokens are placed in place \( P_2 \). Presence of tokens in place \( P_2 \) signifies successful matching of the left-hand side of the rule. A token bearing the rule number 7 is deposited as a token in the place CONFLICT SET. A selection process usually takes place here and if rule 7 is selected the transition RULE7 fires creating a new working memory token with the relation BOSS-DIV. This new token in turn is copied and sent to all places with this name so that a new matching process may commence.
Matched and partially matched tokens reside in places such as $P_1$ and $P_2$, which are usually referred to as the $\alpha$ or $\beta$ nodes in the literature. Presence of these tokens saves on repeated matchings after firing of every rule. It is also possible for rules which share relations to share associated places, so that a single matching process can serve for more than one rule. Such representation can be easily shown in net representation of the intra rule matching process.

VI. MAPPING PROBLEM

A. A New Paradigm

Net theory provides us with a new model for representing parallel algorithms, which is suitable for exploitation of concurrency. One of our objectives in this section is to develop a paradigm for parallel algorithms, which provides a clean separation between various intertwined issues in parallel computation such as:

(i) data-sharing between processes,

(ii) communication relationships between processes,

(iii) causal dependency relationship between processes,

(iv) computational requirements of processes,

(v) hierarchical resolution of an algorithm highlighting different levels of parallelism possible, etc.

In this section we show how net theory based paradigm provides a clean separation between these major issues. Later in the section we show how this paradigm helps us to provide the necessary metrics and measures, which describe the "goodness of fit" between an algorithm and a given architecture. We define the so-called mapping problem
as the problem of allocation of processes to processors in order to minimize the total execution time for an algorithm. We then provide a mathematical formulation of the problem that maximizes concurrency, while minimizing the communication overhead. The formulation will be shown to be NP-complete and various heuristic methods of solution will be suggested. Describing a computation as a directed graph has been an important representational means in the development of parallel computation paradigms. Keller and Davis [24] have proposed a graphical programming language. As noted in [18], visual programming, which is largely graph based, is now an active area of research. Computation graphs [5] also provide a graph-based model of parallel computation. However, none of these graph-based models have been really operationalized on the basis of the graph structure. They have used graphic representation, primarily to explain and explicate their concepts and ideas.

Any model of computation must specify in concrete terms, the basic elements from which the model is derived. A net-theoretic model is based on the idea of a transition being synonymous with an act of computation. An act of computation is accompanied by a change in memory contents. In the net theory paradigm, places constitute memory locations and the tokens constitute contents of the memory. Distribution of tokens over places, i.e., the memory is affected by firing of a transition which constitutes an act of computation. Places that are shared as input and output places by different transitions, constitute shared memory that can be implemented by a shared memory system or by local memory systems with necessary copying of data from one memory to another by message passing between processes. Sharing of places between transitions, thus constitute data sharing and communication between processes.

The communication and the causal relationships between processes are provided by the flow relationships that relate transitions to places and in turn to other transitions. The
computational requirements of a process can be identified with the time taken by a transition. Last, but not least, a net description lends itself to modeling a system at a different level of resolution and detail.

Thus net theory based representation constitutes a powerful paradigm that provides a semantically clean separation between various orthogonal attributes of parallel algorithm, such as concurrency, dependency relationships, communication requirements and other attributes as identified at the beginning of the section. As a paradigm it is conceptually very different, though much more comprehensive, as compared to the popular paradigm of viewing parallel computation solely as a society of communicating processes [22].

B. Metrics

Metrics constitute the necessary means to describe succinctly the communication and the computational characteristics of an algorithm or an architecture. There exists considerable design choices as to the parallel architecture one can choose to fit to a given algorithm. Given such metrics, one can quickly decide on a good architecture with a processor topology, communication network and the memory distribution that would fit a given algorithm. If there are also related metrics to describe an architecture, one could use such metrics to provide a "goodness" of fit or match between an algorithm and the associated architecture. The "goodness" of fit between a class of algorithms and an architecture can provide a capability to predict the performance of such algorithms on the given architecture. The purpose of obtaining "good" fit is to minimize the total computation time.

The total processor time (number of processors * time) for a given computation can be decomposed into time spent for:

(i) Computation.
(ii) **Inter-processor communication.** (We assume in this paper that the time spent in communication between two processors is accounted for by the time to put and receive messages from the network and the time required for a message to travel over the network.)

(iii) **Control tasks and activities.** Such time is generally dependent only on the nature of the algorithm and is invariant how the processes in the algorithm are parcelled out among the processors, which is our central concern here. We therefore neglect time spent over control in our analysis of the mapping problem,

(iv) **Synchronization.** Synchronization is carried out for the purpose of sharing data or for the purpose of control. We include time spent on synchronization for the purpose of data sharing as a part of inter-processor communication. We neglect time spent on synchronization for control for the reason that it is largely invariant to the mapping solution and depends largely on the architecture and the algorithm, both of which are fixed here.

(v) **Idle time due to waiting for messages.** This waiting is due to: (a) the time taken by messages to travel along the network, and (b) due to the process which is supposed to communicate the data whose computation has not been finished yet.

(vi) **Idle time due to non-allocation of processes.** The total processor time can be minimized if we are able to exploit the concurrency available in the system while minimizing overhead due to communication, control and synchronization and also maintaining even loading among the processors. This is easier said than done.
It is obvious that most of the requirements as indicated earlier are mutually contradictory. Exploiting the parallelism in a system by utilizing as many processors as possible increases communication load. In the extreme where communication among processes is very high, optimal mapping in terms of least time for completion of a computation may be achieved by loading all processes in a single processor like a serial processing system. This is also known as the maxi-min problem [28] in the literature on mapping.

There is tradeoff in the mapping process in time lost, due to communication overhead, uneven loading and failure to maximize parallelism in the system. For the sake of simplicity, in this paper we concern ourselves only with factors such as time spent in computation, interprocessor communication, parallelism, even loading, and study how it affects the process of mapping of a given algorithm on to a given architecture. We study how the relation of an algorithm and an architecture can be described in a succinct fashion along each of these conflicting dimensions, by means of suitably defined matrices for a given mapping solution.

Communication Structure: It has only been recently realized that arranging efficient communication between processes is the most critical issue in parallel computation. This is because the time lost in communication could have been devoted to computation and the communication over a network is a much slower process than usual CPU cycle times. The communication structure of a multi-processor system, in the literature, is usually described in terms of diameter, bandwidth, connections between processors, network topology, etc.

We describe the communication structure of a given multiprocessor architecture by a matrix of time $t_{ij}$, which gives the time taken by a processor $i$ to send a packet of data to processor $j$ under conditions of average loading of the network. This includes the time spent by a processor to put a message on the network, time spent by intermediate proces-
sors in the network to route the messages to their destinations, and the time taken by a processor to pick the message from the network. We call it the $T$ matrix. The matrix is of size $N \times N$, where $N$ is the number of processors in the system. The matrix of unit intercommunication times encapsulates important features of an architecture that affects communication time, such as the neighborhood relationship between processors, the efficiency and the nature of the communication network, message passing protocol, etc. For the sake of simplicity, we assume that the time taken by processes to communicate if they are in the same processor is zero.

The net model of an algorithm provides us with the measure of load messages between two processes in terms of token sizes and number of tokens that two transitions need to share. Given such a load matrix from the net model and the interprocessor unit communication time given by the $T$ matrix, one can establish the total processor-time that would be devoted by the system to communication. "Goodness of fit" or match between an algorithm and an architecture with regard to communication relationship, can be provided by the ratio of total time spent in communication in relation to the total computation time.

Assume for a multiprocessor architecture there are 3 different values of $t_{ij}$ in matrix $T$ and which are in a ratio of $t_3, t_2, 1(t_3 > t_2 > 1)$. Let the algorithm mapping be such that the fraction of total message volume is distributed among these three levels as $n_1, n_2$ and $n_3$, where $n_1 + n_2 + n_3 = 1$.

Mapping will be most communication efficient when all messages are exchanged at the most efficient level, i.e., the total processor time (normalized) devoted to communication is equal to 1. In the worst case, all messages are exchanged at the least efficient level, i.e., at unit time $= t_3$ and therefore, the total time devoted to communication will be $t_3$. Thus the measure $\sum t_i n_i$, which varies from 1 to $t_3$, provides a measure of "goodness"
of match between the communication requirement of an algorithm and the communication structure of the multi-processor system for a given mapping. We call this ratio the communication ratio.

As discussed earlier, besides the time spent in communication, there are several other reasons why one may fail to obtain linear speedup in computation. These are: the first is failure to exploit all concurrency in the system. For example, processes that can execute concurrently have been loaded onto the same processor and therefore, execute serially. The second is uneven allocation where we recognize that it is hard to allocate processes evenly. The processors that finish earlier have to wait for processors that finish later. We shall explore in the next few paragraphs, metrics and measures that the net models offer us, which help describe the "goodness" of a mapping of an algorithm on to an architecture in terms of even loading and exploitation of concurrency.

Computation Structure: Net theory provides us with a concept of synchronic distance as a measure of synchrony between two transitions. We will use a modified definition for our purpose. Given a timed net model of an algorithm, we measure synchronic distance between two transitions $t_i$ and $t_j$ by the number of times in the simulation of the net model they fire simultaneously. This can lead to a matrix of values $s_{ij}$ indicating how often transitions $t_i$ and $t_j$ fire together (say for all possible initial markings of the net). We call this the $S$ matrix. Pairs of processes that have high synchronous relationship indicated by high $s_{ij}$ should be allocated across different processors.

A mapping solution is given by a set of indicator variables $z_{mn}$ where $z_{mn} = 1$, if process $m$ is allocated to processor $n$, otherwise it is 0. The product $z_{ik} \cdot z_{jk} \cdot s_{ik}$ is equal to $s_{ik}$ if processes $i$ and $j$ are both allocated to processor $k$, otherwise it is 0. Therefore failure to assign processes according to their synchronous relationship will result in a non-zero product equal to their mutual $s_{ij}$ value. The measure $\left[ \sum_{i>j} \sum_{k=1}^{N} z_{ik} \cdot z_{jk} \cdot s_{ij} \right] / \left[ \sum s_{ij} \right]$ will pro-
vide an indication of how far we have been successful in exploiting the concurrency in the system. The range of this measure is from 0 to 1. We call this ratio the synchronous ratio.

Uneven loading of processes on to processors can be measured by differences of computational loads across processors. If the load on processor \( i \) is \( p_i \), the sum \( \sum p_i^2 \) is minimized when the total load \( \sum p_i \) is distributed evenly across all processors. The ratio \( \left( \sum p_i^2 / N \right) / \left( \sum p_i / N^2 \right) \) is lowest when all processors are equally loaded and can therefore serve as a measure of how evenly the loading has been done in the system. The lowest possible value of such a ratio is 1. We call this ratio the load ratio.

Therefore given an algorithm and a choice of architectures, architectures that have low load ratio and communication ratio and high synchronous ratio in relation to the algorithm, are preferable to architectures which have the opposite characteristics. Based on the above, we can provide the following different metrics and measures of "fit" to describe the relationship of an algorithm and an architecture as obtained by a given mapping.
ARCHITECTURE

COMMUNICATION STRUCTURE

Diameter
Bandwidth
Connections per processor
Topology
$T_{ij}$ matrix

COMPUTATION STRUCTURE

Number of processors
Memory size and topology
CPU cycle time
Number of transitions
$S_{ij}$ matrix
Time for transitions

Figure 14

C. Mapping Problem

For the purpose of mapping a given algorithm to a multiprocessor architecture, algorithms in the literature have been frequently represented as an interconnection graph where the nodes represent processes and the edges represent communication relationships. One of the earliest and influential papers on the subject [4] formulated the problem as a graph isomorphism problem with the number of processors equal to the number of processes. (The two graphs being the interconnection graph representing the algorithm and the graph representing the processor topology.) It also provided an algorithm to map adjacency relationships among processes to network connection between processors. Mapping problems where we not only have the interprocessor network different from the interprocesses network, but also have a difference in the number of processors and the number of processes are considered in [3]. In this paper, the authors describe the first cause of mismatch.
between the network of processes and the network of processors as the *topology* variation and the second cause of mismatch as the *cardinality* variation. Mapping is achieved in two stages. In the first stage, the process network is contracted to the same cardinality as the processor network. In the next stage, the adjacency relationship in the contracted graph is mapped on to the processor network.

The above approaches suffering from a grave lacunae. They all view a distributed process configuration statically as a graphical relationship. The underlying assumption is that all processes in the graph are active at all times and their relationships are fixed. The fact of the matter is far from being this simple. Processes come into existence at different times during the lifetime of a computation and then die away. At different times, different processes are active with their own synchronous and casual relationships across time. The difference in viewpoints between the existing approaches and our approach, could be described as the static versus the dynamic. It is the same difference in viewpoints we came across earlier in Section 2, where we could model a static view of a system by an unmarked place/transition net and we could explore the dynamic behavior of the marked net by studying its reachability space. The net model of a computation helps us to incorporate the dynamic view of the process which is dependent upon the initial conditions, which in this case correspond to the initial marking.

The net model allows us to adopt a dynamic view of the process where different transitions are alive at different times, and it describes precisely these relationships as to causality and synchrony. Our approach to the mapping problem would be to map processes to processors in such a way as to minimize the joint weighted costs due to the following: (i) uneven leading, (ii) poor communication fit, and (iii) allocation of processes without concern for the synchronous relationships. These costs can be weighted according to the judgement as to which factors would be predominant in determining the efficiency
of the computation.

We provide two different mathematical formulations of the mapping problem. The first is based on zero-one integer programming. The formulation will employ a large number of variables and equations, therefore it is not computationally tractable. The second formulation provides a succinct description of the problem, but only on the basis of several simplifying assumptions. In the second formulation, the problem is described as an quadratic assignment problem, and different heuristic methods for solutions to such problems are considered.

**Model 1:**

The purpose of the present section is to provide a formal model of the mapping problem with as few assumptions as possible. Consider a mapping problem where pre-emption of processes from processors is disallowed and we follow the notations as used in [51], a paper on resource-constrained project scheduling.

The solution to the mapping problem in this model determines what processes are to be allocated to what processors and when. The objective of the model is to find a time-based mapping, such that the total computation time is minimized. We assume that a net representation of the computational process is available. The net representation helps us to derive the data-flow diagram representing this computation. Given a net representation and rules for resolution of conflicts in the net, a data-flow diagram can be easily obtained.

It is pertinent to note here the similarities of these two representational models, i.e., the data-flow method and the Petri net representation. Both methods take a 'side-effect' free and a functional viewpoint of computation. In dataflow representation, computation is driven by the arrival of data to the input of the functions. Like any functional language paradigm such as LISP, CLU, etc., the results of the computation is solely dependent on
the input data and is free of the environment in which the function is computed. In net
representation, the computation is driven by arrival of tokens in the input places of the
transitions and the result of the transitions are dependent solely on the token attributes.

A functional and a "side-effect" free viewpoint of a computation is necessary for
distributing a computation into several concurrent processes which then have only a single
well defined relationship, which is the precedence relationship. No other restrictive rela-
tions need be assumed. It is therefore no accident that we begin with a dataflow representa-
tion of the problem for parallel implementation. It should be noted that a data flow
diagram can be easily derived from a net representation, but the opposite is not true since
such a diagram does not specify how the conflicts are resolved in the net. In that sense, a
net representation of a computation is far more general.

A dataflow diagram can also be viewed as an acyclic activity-on-node graph where
activities are the nodes and the directed arcs show the flow of data from one activity to
another. Therefore the directed arcs also specify the precedence relationships among
activities. The nature of the diagram is the same as the CPM (Critical Path Method)
activity network representation of a project.

In keeping with the general nature of the model, we have included two constrained
resources: processors and copies of functions. The various computational processes as
shown in the dataflow diagram are functions, some of which are common. We assume
that the copies of such functions are limited and have to be assigned to processors at the
beginning of the computation. Copies of such functions cannot migrate from one proces-
sor to another. Problems in mapping with limited copies of functions has been previously
investigated in [28].

The general time-resource tradeoff problem can be formulated as an integer program-
mimg problem in which a zero-one integer variable $x_{ij} = 1$ if process $i$ is assigned to
processor \( j \) and is scheduled to be over in time \( t \). Processes are labeled from 1 to \( n \), with \( n \) as a terminal dummy process with no resource requirement and which takes zero time. Associated with each process \( j \) is the critical path determined by the early finish time \( E(j) \) and by the late finish time \( L(j) \). Readers may consult [52] for methods for calculating early and late finish time for an activity based on an activity network. For calculating the late finish time \( L(n) \) is set equal to a high value such as the sum of all processes duration times.

We assume that the following data is available to us:

1. The duration of each process. It is assumed that the processors are identical and all processes can be done on all processors. The duration of a process \( j \) is \( d(j) \).

2. The quantum of dataflow from one process to another in terms of datapackets. The dataflow requirement from process \( j \) to process \( i \) is given by \( m_{ji} \).

3. The \( T \) matrix that gives us the time taken by a unit data packet to flow from one processor to another. The time taken by a data packet to flow from processor \( i \) to processor \( j \) is given by \( T_{ij} \).

4. The set of functions or procedures \( \{ f_1, \ldots, f_k \} \) and the knowledge which process corresponds to each function. This is given by integer variable \( V_{jk} \) which is set equal to 1 if process \( j \) is function \( k \), otherwise it is set to 0.

**Objective function.** Minimize:

\[
\sum_{j=1}^{N} \sum_{i=E(n)}^{L(n)} t \cdot X_{nij}
\]

such that

\[
\sum_{k=1}^{N} \sum_{i=E(j)}^{L(j)} X_{jk} = 1
\]
for each $j = 1, \ldots, n$ processes (a process can be assigned to a processor to finish at one
time only).

$$X_{a,j} = \sum_{t \in E(a)}^{L(a)} X_{a t j}$$  \hspace{1cm} (3)

for all processes $a = 1, \ldots, n$ and all processors $j = 1, \ldots, N$ (if $X_{a,j} = 1$, then process
$a$ has been assigned to processor $j$).

$$X_{a t} = \sum_{j=1}^{N} X_{a t j},$$ \hspace{1cm} (4)

for all processes $a = 1, \ldots, n$ and all $t = 1, \ldots, H$ (if $X_{a t} = 1$, then process $a$ is
scheduled to finish at time $t$).

$$\sum_{i=1}^{n} \sum_{q=1}^{(t + d(t)) - 1} X_{i q j} \leq 1_i$$ \hspace{1cm} (5)

for all time $t = 1, \ldots, H$ and processors $j = 1, \ldots, N$ (at any one time only one process
at most can be assigned to only one processor $1_i = 1$ for all $i$).

$$W_{kl} \geq V_{k l} \cdot V_{j l},$$ \hspace{1cm} (7)

for all functions $k = 1, \ldots, K$ and processors $l = 1, \ldots, N$. $W_{kl}$ is 1 if function $k$ is
assigned to processor $l$, otherwise it is set to 0. (A process $j$ of function type $k$ can be
assigned to processor $l$ if that function type has been assigned to that processor.)

$$\sum_{i=1}^{N} W_{kl} \leq C_k$$ \hspace{1cm} (8)

for all function types $k = 1, \ldots, K$ (the number of copies of a function $k$ that can be dis-
tributed is limited by number $C_k$).

$$- \sum_{i = E(a)}^{L(a)} t \cdot X_{a t j} + \sum_{t = E(b)}^{L(b)} (t - d (b)) \cdot X_{b t l} \geq$$

$$m_{ba} \cdot T_{j l} - (1 - X_{a t j}) \cdot M - (1 - X_{b t j}) \cdot M , \ldots,$$ \hspace{1cm} (9)

for all pairs $(a,b)$ which are elements of the list $P$, and for all processor pairs $(j,l)$ where
$j,l = 1, \ldots, N$ (The list $P$ has pairs of all processes which have immediate precedence
relationship defined by the dataflow diagram. The equation enforces the relationship that
before process \( b \) can commence process \( a \), the predecessor must finish and the time required for the data to move from processor \( j \) to processor \( l \), must elapse. If the processes \( a \) and \( b \) are not scheduled on processors \( j \) and \( l \) respectively, the presence of a large number \( M \) in the equation disenables the relationship.) From the above model, it is evident that the number of variables is in the order of \( O(n,H,N) \), i.e., the product of the numbers of processors, processes and time. Number of equations of type (6) is in the order of \( O(n^2,N^2,H) \). Such a description of a problem is therefore far from tractable. In the next model, we will provide a more parsimonious description of the problem where we will exploit the properties of the net description of the problem in order to incorporate the important elements required for the allocation process.

We formulate the model as follows:

(i) Let the set of processes be \( X = \{x_1, \ldots, x_n\} \).

(ii) Let the set of processors be \( P = \{p_1, \ldots, p_N\} \).

(iii) Variables \( x_{ij} = 1 \), if process \( i \) is allocated to process \( j \), otherwise 0.

(iv) DATA: (i) \( S \) matrix = \( (s_{ij}) \). The matrix is of size \( n \times n \) and gives the synchrony relationship between processes \( i \) and \( j \). (ii) \( T \) matrix = \( (t_{ij}) \). The matrix is of size \( N \times N \) and gives the unit time to transmit a data packet from processor \( i \) to processor \( j \). (iii) \( L \) matrix = \( (l_{ij}) \). The matrix is of size \( n \times n \) and gives the communication load from processes \( i \) to \( j \). (iv) \( C \) vector = \( (c_i) \), is of size \( n \) and represents the computational time requirement of a process \( i \).

(v) Constraints:

\[
\sum_{j=1}^{N} x_{ij} = 1, \quad (i = 1, 2, \ldots, n) \tag{1}
\]

a process can be allocated to only one processor.
\[ \sum_{i} c_{i,j} \cdot x_{ij} = z_{j}, \quad (j = 1, \ldots, N) \] (2)

\( z_{j} \) is the load allocated to processor \( j \).

(vi) Objective function \textsc{Minimize}

\[ \alpha \cdot \sum_{i > k} \sum_{i = 1}^{n} \sum_{j = 1}^{N} x_{ij} \cdot x_{kj} \cdot s_{ik} \quad \text{[synchronous costs]} \]

\[ + \beta \cdot \sum_{i > k} \sum_{j = 1}^{N} \sum_{k = 1}^{n} \sum_{l = 1}^{n} x_{ij} \cdot x_{kl} \cdot t_{jl} \cdot l_{ik} \quad \text{[communication costs]} \]

Quadratic Assignment Problem: The above is a general form of the quadratic assignment problem that is discussed in Operations Research literature. Quadratic assignment problems have a very broad range of application such as in design wiring layout, route selection, facility location problems, etc. Interpreted in the context of assignment of processes to processors, there is usually \( n \) processes to be assigned to \( N \) processors. There is an affinity relationship between the processors which is given by the \( N \times N \) matrix, an affinity relation between processors which is given by the \( n \times n \) matrix and a relationship between processor and processes, which is given by a \( N \times n \) matrix. These affinity relationships are in the nature of 'costs' and the problem is to find an assignment that minimizes these costs.

Consider the quadratic assignment problem in terms of mapping a set \( M = \{1, \ldots, m\} \) into a set \( N = \{1, \ldots, n\} \) where \( m > n \). Each element of \( M \) is assigned to one distinct element of \( N \) and more than one element of \( M \) can be assigned to a single element of \( N \).

Let \( S \) be the set of all possible mappings of the set \( M \) into set \( N \). A particular mapping \( \rho \in S \) can be represented in the form

\[ \rho = i_{1}, i_{2}, \ldots, i_{m} \quad i \in M \]
\[ j_{1}, j_{2}, \ldots, j_{m} \quad j \in N \text{ where } \rho(i_{k}) = j_{k} \]
The quadratic assignment problem consists of finding, from among a set of possible alternative solution set $S$, a map $\rho$ that optimizes the value of objective functions.

Algorithms for producing an optimum solution exist. These are standard algorithms for solving integer programming problems such as branch-and-bound etc., but are not computationally feasible for problems of the size exceeding 10 locations. The quadratic assignment problem has been proven to be an NP-Complete problem [42] and is therefore not likely to have polynomial solution. It can be easily reduced to the mapping problem that we have considered and therefore it is trivial to prove that the mapping problem is also NP-Complete.

The heuristics methods which have been developed up to now can be classified into (i) constructive initial placement techniques, and (ii) iterative improvement techniques. The constructive technique is an $n$-stage decision process for intelligently building a solution from scratch. It was developed by Graves and Whinston [17]. It uses a general enumerative procedure based on probability theory to form an implicit algorithm. It was later revised [30] to include a back-tracking strategy for generating alternative solutions. Iterative techniques attempt a hill-climbing strategy where at every iteration the solution is incrementally improved. Among the iterative techniques the one developed by Hillier [19] has proved to be efficient in many different situations. It was later revised [38]. A new and very different approach to solving mapping problems is the so-called simulated annealing technique [25].

Simulated annealing technique is a probabilistic modification of the traditional neighborhood search technique or the hill-climbing technique. In hill-climbing techniques, only solutions which improve the value of the objective function are considered. In simulated annealing, the decision criteria are more complex by virtue of assigning a non-zero probability of selecting a solution that is worse than the existing solution. This probability is
dependent on a parameter called the temperature. At higher temperature, the search is more random and therefore more likely to avoid getting locked into any local optimum. As the temperature is gradually reduced, implying that randomness of the search technique is reduced, the process moves closer to being a usual neighborhood search technique. Simulated annealing has been used for the mapping problem [2].

VII. KNOWLEDGE ENGINEERING

Commercial exploitation of expert systems has become a phenomenon in the nineties. However, there is still a large gap between the number of prototypes being developed and being pressed into service. Knowledge engineering has proven to be a bottleneck in the building of prototypes and their ultimate placement into regular use.

Knowledge engineering is the process by which expertise is obtained from experts and a rule base, if formulated so that an expert system shell can process the rule base. This activity is generally viewed as consisting of three interrelated and intertwined processes: (1) knowledge acquisition, (2) rule based design and building, and (3) validation and implementation. The process of capture and implementation of expertise in a software form is largely an ad-hoc exercise, even though many techniques and methodologies are being investigated. In the area of capture, several techniques have shown promise such as protocol analyses, expertise transfer systems, psychometric techniques, etc. [46]. However, there is little achievement in the design of rule based systems (after the expertise has been obtained from experts) and their validation. It is in the area of rule based design and validation that most expert systems under development flounder.

The enormity of the task of building up an expert system can be gauged by the fact that a real life expert system usually has thousands of rules and involves several programmers over a period of months. R1 has a rule base of 2153 production rules, XSEL has a rule base of 1303 rules [34], and XCON, which started several years ago with a rule base
of seven hundred, has currently about 6200 rules of which about 50% is changed yearly to cope with model changes in computers [43]. This also illustrates the importance of maintainability of expert systems, once a system has been brought into existence. Due to changes in the problem domain, easy maintainability of a system is necessary to cope with the problem of "integrity-degradation" over time. As the number of rules in a system grow and different programmers work on the same rule base with different understandings of what rules do or mean, it is beyond the scope of any one person or even a small group to visualize how a rule set of a few thousand rules interact. It is common for programmers to employ "tricks" in order to specify relationships among rules over and above to what is allowed in the expert system paradigm [43]. It is not always possible for other programmers to decipher several years down the line what these rules or "tricks" were meant to achieve. At a later stage, when new rules are being inserted, it becomes impossible to prevent these from interacting in unforeseen and undesirable ways with the old rule set.

The present chaotic and ad-hoc state of the subject of knowledge engineering, bears a strong resemblance to the state of programming in the early seventies before the era of structured programming. There is an acute need for such a structured methodology in the subject of knowledge engineering. Structured programming brought to software engineering, a metalanguage for describing a program (which is the top-down paradigm) and showed how a few control schemes such as if/then, begin/end and while/do could bring about order and discipline in the jungle of different control structures.

There is a similar need in knowledge engineering for a methodology that can bring along a simple and universal grammar for describing any expert system and help formulating its rule base. Any such methodology must have the ability to describe an expert system unambiguously, in precise terms so that different programmers working on the same
description of the expert system would design systems with identical properties. For such a methodology to prove practical, it must be able to describe systems in various levels of detail and resolution.

Hierarchical resolution is a tool used to reduce the conceptual complexity of a system. It acts as an aid to comprehension and understanding. Hierarchy provides a controlled method of selectively hiding or exposing the details of a complicated system. Experts will generally describe a given system at various levels of resolution. Any method for structuring an expert's knowledge must be easily amendable to such hierarchical description. Such a method should also serve the need for referential transparency.

Referential transparency implies that a system can be described in terms of some basic processes without concern for how the processes are implemented. It helps in breaking up a large task into smaller parts and delegating each part to separate programmers who need not interact among themselves to produce their individual parts. This permits delegation of work, control and evaluation of each programmer's output in isolation and later easy assembly of individual outputs.

A. Knowledge-base Development

Petri net theory with its simple set of model primitives such as places, transitions, and tokens provide an uncomplicated and a universal grammar for describing expert systems for any application. Flow relations in a net representation serves to fix in exact terms the dynamic behavior of a system and, in the case of expert system, how the various rules happen to relate to one another. Net representation by its 2-dimensional syntax can provide a graphical representation of an expert system that can also serve the same purpose as an engineering drawing does in the domain of tolerance-controlled manufacturing. It is precise and is free from any contextual and unspoken understandings. As an analogy different manufacturing shops working on the same drawing come out with identical
products.

Net theory is well suited to hierarchical resolution. At the most extreme, a whole expert system can be described by a single transition with a single output and input place. The input place describes the data that is input into the system and the output place describes the data that is output from the system. This system description is devoid of any detail as to how that transition is brought about and thus supports the principle of referential transparency. This single transition only provides the relation between the input data and output data and serves as a black-box description of the transition solely as it relates to the world outside the transition itself.

A single transition as above can be exploded into several transitions that would describe the same transition in terms of other more primitive transitions. Each transition at this higher level of resolution is again specified in terms of its input/output places, i.e., in terms of its input and output data. Thus we achieve the description of a black box at one level in terms of black boxes as greater resolution in the next level of detail. This process of increasing resolution can continue until the system gets described in terms of primitive transitions that can be directly expressed in the language supporting the development of the expert system.

The net description of a system being in a graphic form can be studied easily by the expert as well as the knowledge engineer. "What-if analyses" can be conducted easily on such a system description by studying its dynamic behavior. This provides both the expert and the knowledge engineer a common language to describe the system and study the consequences of the relationships among data and rules as described by the expert. Such a description of a system may help an expert gain insight into his own expertise and help improve it. The basic elements of a net grammar are simple and it may be possible for an expert to describe an expert system directly in a net form without the intervention of any
knowledge engineer. This representation can serve as a hard copy description of his expertise at a particular point of time. The effort of building up a net representation will induce the expert to focus on key relationships, important data, relationships among relationships, etc.

B. Rule Base Validation

In the next few paragraphs, we will demonstrate how the net representation and its underlying graph can be analyzed to provide indications as to the presence of contradictions, redundancy, cycles, disconnected components and the like in the rule base.

(1) Disconnected Components: It is easy to check for disconnected components by analyzing the underlying graph of the net. The underlying graph of the net consists of all the places and transitions in the net represented as vertices with the existing arcs joining them. Algorithms exist that can identify disconnected components in the order of $O(|\text{edges}| + |\text{vertices}|)$ [47].

(2) Graph Partition: To study a net representation of a large system, it is necessary to partition the representation in a meaningful fashion. One would prefer partitions of a net such that the elements of each partition relates strongly among themselves (in terms of edges) in contrast to elements with other partitions, i.e., each partition should form some sort of a “clump”. Partitioning is also important from the point of view of documentation and maintainability of an expert system. Graph partitioning is a member of NP-complete problem [1]. Several heuristic methods exist for graph partitioning. A recent one called “Stochastic iterative genetic hill-climbing” (SIGH), has demonstrated good performance over a wide range of random graphs [1].
(3) **Redundancy:** When rules number in thousands in a rule base and undergo modifications every year, it is not trivial to detect if the new rules being inserted already exist in the rule base. A simple method for checking rule redundancy can be based on a "Godel numbering" scheme. A transition in a net is described in terms of the input places, output places and inscriptions, if any. The places and transitions in a net are identified as a member of a set of some ordered indices. These indices could be natural numbers and the inscription could also be converted to natural numbers. A transition under this scheme would then be uniquely identified as a single though large number, which is the concatenation of the identities of its input places, output places and the inscriptions as numbers. These identity numbers can be sorted in the order $O(n \log n)$ and in another order $O(n)$ steps the ordered set could be checked for duplication of numbers. This does not take into account the fact that some numbers will be very large. Duplicate numbers in the ordered set would indicate duplication of rules. Adjacent numbers, if they are very close, could also be checked if the transitions represented by these numbers differ in a meaningful way.

(4) **Contradiction:** As discussed earlier contradictions and inconsistencies are common in expert systems. It is both a strength and weakness of this method. It is a strength because it provides a richer repertoire of behavior and it is a weakness because it makes the behavior of the system unpredictable. A variable in an expert system would generally bind itself to different values during the run-time. This however, may not be desirable in some situations and can be viewed as providing a contradictory or inconsistent behavior. For a net representation, this would be exhibited by tokens with different values or attributes getting generated at the same place. This can be detected by methods similar to those described in Section V for identifying contradiction in a set of clauses.
(5) **Cycles:** Readers may consult [16] for a method for identifying cycles in the underlying acyclic graph of a net representation of a system. The method has the order $O(|\text{edges}| + |\text{vertices}|^3)$. Once cycles have been identified in the underlying graph, these cycles can be checked for cycles of transition. Another method for checking cycles of transition as much as firing of transitions beyond an order of $O(|\text{places}| + |\text{transitions}| + |\text{arcs}|)$ would indicate transition cycles in the net.

**Conclusion**

The primary goal of the paper is to explore the potential for representing deductive systems found in logic and expert systems. By establishing a representation so that deductive inference can be expressed in net theory concepts, AI systems can be analyzed using the full power of net theory. Techniques associated with net theory such as the construction of reachability sets, transformation methods based on net morphisms and invariant analyses can be used to study inconsistencies and contradictions in the knowledge base, to detect deadlocks and to study issues relating to redundancy.

Net formalism also provides a means for graphical representation of a system. It furnishes a machine-independent, programming language free computational description of deductive algorithms. This has an important consequence in that descriptions of large interacting AI systems are readable and can be easily adapted to different programming and software environments.

Net models are essentially flow models of computation. They make explicit the different levels of parallelism inherent in a given computation. We have shown how timed net models of computation can be used to map algorithms to different multi-processor architectures so that the concurrency available in the computation can be exploited. We have also shown that adaptation of colored nets to represent AI systems gives a
parsimonious yet effective method for representing computation. It short, net computation models of AI systems provide us with means for graphical representation, system analyses and validation and as a model for implementing AI algorithms in distributed processing environments.

There are other potential areas of research in net based computational approach that may prove rewarding. Future investigations will seek computationally efficient methods for conducting inference in non-horn type clausal reasoning systems, and also efficient methods for validation of knowledge bases. Net methods are well suited for hierarchical representation of large systems. Future research will focus on using net representation for analyses of large aggregate knowledge-based systems and for representing non-monotone belief structures.

Literature


(1985).


