Proof Procedure and Answer Extraction in Petri Net Model of Logic Programs

GEORGE PETERKA AND TADAO MURATA, FELLOW, IEEE

Abstract—**This paper discusses a proof procedure and answer extraction in a high-level Petri net model of logic programs. The logic programs are restricted to 1) Horn clause subset of first-order predicate logic, 2) finite problems. The logic program is modeled by a high-level Petri net and the execution of the logic program or the answer extraction process in predicate calculus corresponds to a firing sequence which fires the goal transition in the net. For the class of the programs described above, the goal transition is potentially firable if and only if there exists a nonnegative T-invariant which includes the goal transition in its support. This is the main result proved in this paper. Three examples are given to illustrate in detail the above results.**

Index Terms—**Answer extraction, Horn clause, invariant, logic programming, Petri nets, theorem proving.**

I. Introduction

THE introduction of logic programming has opened a new era for computer science because it provides a uniform formalism for diverse aspects of computer science, especially in artificial intelligence. Logic or logic programs provide us with a general purpose problem-solving language, a concurrent language suitable for operating systems and also a foundation for deductive database systems [1], [2].

This paper develops a model of logic programs for proof procedures and answer extraction. The main motivation of this work is to provide new insights and strategies for concurrent processing of logic programs and automatic theorem proving. The method is based on converting the logic program to a high-level Petri net and then determining the potential firability of the transition corresponding to the goal clause of the program. It turns out that for Horn-clause logic programs the goal transition is potentially firable if and only if (iff) there exists a nonnegative T-invariant which includes the goal transition in its support. Since techniques for finding T-invariants in high-level nets exist, this method may have practical applications.

Manuscript received October 31, 1986; revised October 30, 1987. This work was supported by the National Science Foundation under Grant DMC-8510208.

G. Peterka was with the Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, Chicago, IL 60680. He is now with Illinois Advanced Design Corporation, 1251 South Wolf Road, Hillside, IL 60162.

T. Murata is with the Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, Chicago, IL 60680.

IEEE Log Number 8825086.

A method for modeling logic programs by high-level Petri nets was presented by Murata and Zhang [3]. Lautenbach [4] found a necessary and sufficient condition for a set of clauses to contain a contradiction based on analysis of the Petri net model of the set of clauses. Lautenbach's condition turns out to be false for non-Horn clauses [14], but it can be adapted for Horn clauses. Other earlier related work is listed in the references of [3] and [15].

For brevity, it is assumed that the reader is familiar with basic terminology in logic programming and automatic theorem proving [1], [2].

II. Preliminaries

A *Petri net* [5], [6] is defined as the 5-tuple $N = (P, T, E, M_0, W)$. P and T are two disjoint sets of vertices called *places* and *transitions*, respectively. E is the set of *arcs* consisting of directed edges from a place to a transition or from a transition to a place. M_0 is a function from P to the set of nonnegative integers and denotes the initial token distribution, called the *initial marking*. W is a function from E to the set of nonnegative integers and $W(e)$ denotes the *weight* or multiplicity assigned to arc e. In this paper, all places have infinite capacity.

A transition is said to be *enabled* or *firable* if each of its input places has at least as many tokens as the weight of the respective incoming arc to the transition. An enabled transition may or may not fire. When an enabled transition fires, the number of tokens in each of its input places decreases by the weight of the respective incoming arc to the transition and the number of tokens in each of its output places increases by the weight of the outgoing arc from the transition.

A bag, or multiset, is an extension of a set which allows multiple occurrences of elements [5]. For example, $B = \{a, c, c\}$ is a bag over the domain $\{a, b, c\}$ with the number of occurrences, $\#(a, B) = 1$, $\#(b, B) = 0$ and $\#(c, B) = 2$. An ordered bag is called a *sequence* and is denoted by $< \ >$. A sequence σ can be converted to a corresponding bag B by $B = \text{Bag}(\sigma)$. For example, for the sequence $\sigma = < c, a, c >$, we have $B = \text{Bag}(\sigma) = \{a, c, c\}$. The concatenation of two sequences σ_1 and σ_2 is denoted by $\sigma_1 \cdot \sigma_2$. A firing sequence is a sequence of transitions.

The *Parikh mapping* of a bag B is denoted by $\psi(B)$ and is defined by

$$\psi(B) = \left(\#(t_1, B), \#(t_2, B), \cdots, \#(t_n, B)\right)$$

where $\{t_1, t_2, \cdots, t_n\}$ is the domain for B. $\psi(\sigma)$, the Parikh mapping of a firing sequence σ, is similarly defined and is denoted by $\bar{\sigma}$. A firing sequence $\sigma = < t_1, t_2, \cdots, t_n >$ is said to *transform* a marking M_0 into a marking M_n. This can be denoted by any of the following ways:

$$M_0[\sigma > M_n$$

$$M_0[t_1, t_2, \cdots, t_n > M_n$$

$$M_0[t_1, M_1, t_2, M_2, \cdots, t_n > M_n$$

where M_i is the marking which results after the ith transition in the firing sequence fires. A firing sequence $\sigma = < t_1, t_2, \cdots, t_n >$ is said to be *executable from* M_0 if t_1 is firable from M_0, and t_2 is firable from M_1, and so on for all transitions in σ. A transition t is said to be *potentially firable* if it can be made firable through some firing sequence.

The *indegree* (*outdegree*) of a transition t is an integer equal to the sum of the weights of all incoming (outgoing) arcs of t. A transition is called a *source* (*goal*) if it has indegree (outdegree) = 0.

The *incidence matrix* for a Petri net with n transitions and m places, $A = [a_{ij}]$ is an $n \times m$ matrix of integers. a_{ij} is the weight of the arc from transition i to place j minus the weight of the arc from place j to transition i. An n-vector of integers, X, is called a *T-invariant* if $A^T X = 0$. The ith entry of a vector X is denoted by $X(i)$. A subset of transitions corresponding to nonzero entries of an n-vector $X \geq 0$ is called the *support* and is denoted by $\| X \|$. A T-invariant, $X \geq 0$ is said to be *executable from* M_0 if there exists a firing sequence σ executable from marking M_0 such that its firing count vector $\bar{\sigma} = X$. The following lemma regarding T-invariants is well known [6].

Lemma 1: An n-vector $X \geq 0$ is a T-invariant iff there exists a marking M_0 and a firing sequence σ from M_0 back to M_0 such that $\bar{\sigma} = X$.

Lautenbach's *net representation* [4] N of a set of clauses R in propositional logic is defined to be a Petri net $N = (P, T, E, M_0, W)$ where:

1) The set of transitions T equals the set of clauses R.
2) The set of places P is the set of all occurring literals in nonnegated form.
3) The set of arcs $E \subseteq P \times T \cup T \times P$ is given by

$$(p, t) \in E \cap (P \times T) \text{ iff } \neg p \text{ occurs in } t,$$

$$(t, p) \in E \cap (T \times P) \text{ iff } p \text{ occurs in } t.$$

4) The initial marking $M_0 = 0$.
5) The weight function W is given by

$$W: E \to \{1\}.$$

Colored Petri nets [7] and predicate-transition nets [9] are two types of high-level Petri nets. It is known that these two high-level Petri nets are equivalent in the sense that any concept, algorithm or theorem which applies to one approach also applies to the other [11]. In this sense, the two types of nets are interchangeable. The formal procedure for converting a logic program into a high-level Petri net is presented in [3]. An informal method for converting a logic program into a high-level Petri net is given and is illustrated by an example in Section IV in this paper.

The following lemma has been proved in [15] and is used to prove Theorem 2 in the present paper.

Lemma 2: Given a Horn-clause logic program P and a goal G, we have the high-level Petri net representation Ω for P and a goal transition t_g for G. There exists a contradiction in $P \cup \{\neg G\}$ iff t_g is potentially firable from the empty marking in Ω.

It is assumed in this paper that logic programs are restricted to Horn-clause predicate logic problems whose high-level Petri net representation contains a finite number of colors and thus can be unfolded into a finite uncolored or ordinary net. This is essentially a restriction to finite problems.

III. Main Results

Theorem 1: Let $N = (P, T, E, M_0, W)$ be a Petri net having all transitions with outdegree ≤ 1. Let t_g be a goal transition in T. There exists a firing sequence which reproduces the empty marking and fires the goal transition t_g in N iff N has a T-invariant X such that $X \geq 0$ and $X(t_g) \neq 0$.

(The following proof is highly technical and thus the casual reader may wish to skip this proof for the first reading.)

Proof: The necessity is obvious since if such a firing sequence exists, its firing count vector is a T-invariant $X \geq 0$, $X(t_g) \neq 0$.

The sufficiency can be proved as follows. Since N has a T-invariant $X \geq 0$, by Lemma 1 there exist a marking M_0 and a firing sequence σ from M_0 back to M_0 such that $\bar{\sigma} = X$. Let B be the bag of transitions such that $\psi(B) = X$. Let σ' be any firing sequence satisfying the following four conditions:

1) $B' = \text{Bag}(\sigma')$.
2) $B' \subseteq B$ (B' is a subbag of B).
3) $0[\sigma' > M$, where 0 is the empty marking.
4) No transitions $t \in (B - B')$ are firable from M.

The following is a proof by contradiction that $t_g \notin (B - B')$.

Assume that $t_g \in (B - B')$.

$\sigma'' = \sigma' \cdot t_g \cdot \sigma_1$, where $\text{Bag}(\sigma_1) = (B - B' - \{t_g\})$

$$\begin{aligned}\text{Bag}(\sigma'') &= \text{Bag}(\sigma') + \text{Bag}(t_g) + \text{Bag}(\sigma_1) \\ &= B' + \{t_g\} + (B - B' - \{t_g\}) \\ &= B\end{aligned}$$

$\bar{\sigma}'' = \psi(\text{Bag}(\sigma'')) = \psi(B) = X$, $\therefore M_0[\sigma'' > M_0$

$M_0[\sigma', M_1, \{t_g\}, M_2, \sigma_1 > M_3 = M_0$

$M_1 = M_0 + M$

$\exists p \in P, M_2(p) < M_0(p)$

This is true because of the following: t_g is not firable from M because we assumed that $t_g \in (B - B')$. Since t_g is firable from $M_1 = M_0 + M$, at least one of the tokens from M_0 must be removed by firing t_g resulting in a place p such that $M_2(p) < M_0(p)$.

$$\exists p' \in P,\ M_3(p') < M_0(p')$$

This is true for the following reason: no transitions in σ_1 are firable from M. Firing any transition will remove at least one token from M_0 and will add at most one token because outdegree ≤ 1. Therefore, there will always exist at least one place p' such that $M_3(p') < M_0(p')$.

$\therefore M_3 \neq M_0$ and a contradiction has been reached.

$t_g \notin (B - B') \Rightarrow t_g \in B'$, i.e., t_g is potentially firable.

Let σ''' be a firing sequence satisfying the following three conditions:

1) Bag $(\sigma''') \subseteq B'$.
2) $0[\sigma''' > 0$.
3) $t_g \in \text{Bag}(\sigma''')$.

Since all transitions have outdegree ≤ 1, σ''' exists and can be constructed by repeatedly removing the last occurrence of any transition from σ' which deposits a token into a place p'', $M(p'') \neq 0$. By Lemma 1, since σ''' reproduces the empty marking, $\bar{\sigma}''' = X'$, $X' \geq 0$, X' is a T-invariant.

Corollary 1: $\|X'\| \subseteq \|X\|$.

Proof: Bag $(\sigma''') \subseteq B$.

Intuitive Explanation of Theorem 1: Since each transition can create at most one token, the net has the following interesting property. Tokens are created by the source transitions, propagated along by the interior transitions, and removed by the goal transition. This can be viewed as a flow of tokens from the sources to the goal. All of the tokens in this flow are removed when they reach the goal.

Since $X \geq 0$ is a T-invariant, by Lemma 1, there exists a marking M_0 and a firing sequence σ executable from M_0 which reproduces M_0. Any tokens in M_0 must remain at the completion of σ. The tokens in M_0 cannot participate in the flow of tokens to the goal because all of the tokens in this flow are removed. The flow of tokens to the goal is totally independent of the tokens in M_0 and thus can occur from $M_0 = 0$. This flow corresponds to the firing sequence σ' which is executable from $M_0 = 0$ and reproduces $M_0 = 0$. By Lemma 1, $X' = \bar{\sigma}'$ is a nonnegative T-invariant.

We can relax the condition of the empty marking reproducibility in Theorem 1, and state the following corollary.

Corollary 2: Let $N = (P, T, E, M_0, W)$ be a Petri net having all transitions with outdegree ≤ 1. Let t_g be a goal transition in T. t_g is potentially firable from $M_0 = 0$ iff N has a T-invariant X such that $X \geq 0$, $X(t_g) \neq 0$.

Proof: The sufficiency is proved in the proof of Theorem 1. The necessity can be proved as follows. Let $\sigma = < \cdots t_g >$ be a firing sequence such that $0[\sigma > M$. If $M = 0$, then $\bar{\sigma}$ is a desired T-invariant $X \geq 0$, $X(t_g) \neq 0$. If $M \neq 0$, then a firing sequence σ' such that $0[\sigma' > 0$ and $t_g \in \text{Bag}(\sigma')$, can be constructed from σ by repeatedly removing the last occurrence of any transition which deposits a token into a place p, $M(p) \neq 0$, since all transitions have outdegree ≤ 1. By Lemma 1, since σ' reproduces the empty marking, $\bar{\sigma}' = X$, $X \geq 0$ is a T-invariant.

Corollary 3: Let $N = (P, T, C, M_0, W)$ be a high-level Petri net having all transitions with outdegree ≤ 1 and a finite number of colors, $|C| < \infty$. Let t_g be a goal transition in T. There exists a firing sequence which reproduces the empty marking and fires the goal transition t_g in N iff N has a nonnegative T-invariant X such that $X(t_g) \neq \varnothing$.

Proof: A high-level Petri net N, with a finite number of colors, can be unfolded [7], [8] into an equivalent finite uncolored or ordinary Petri net N_u. Each place p is unfolded into a set of places (one of each kind of tokens which p may hold). Likewise each transition t is unfolded into a set of transitions (one for each way in which t may fire). If a transition has outdegree ≤ 1 in N, then each of the transitions into which it is unfolded in N_u will have outdegree ≤ 1. The T-invariant $X \geq 0$ is unfolded into a set of nonnegative T-invariants in N_u. Since $X(t_g) \neq \varnothing$ there will exist at least one T-invariant X' in the uncolored net with $X'(t_g) \neq 0$. Theorem 1 holds for X' and produces a firing sequence which reproduces the empty marking. Therefore Corollary 3 follows from Theorem 1.

Theorem 2: Let $N = (P, T, E, M_0, W)$ be the net representation of a finite set R of Horn clauses with finitely many literals. Let R contain one goal clause. R contains a contradiction iff there exists a T-invariant $X \geq 0$, $X(t_g) \neq 0$, where t_g is the goal.

Proof: All transitions in N will have outdegree ≤ 1 because of the Horn clause form. Since R has exactly one goal clause, N will have exactly one goal transition t_g. The proof follows directly from Lemma 2 and Corollary 2.

Remarks: The major difference between Lemma 2 and Theorem 2 is that Theorem 2 requires only that a nonnegative T-invariant, X which includes the goal transition in its support be found. Lemma 2 requires determining the potential firability of the goal transition. (Reference [14], which came to the authors' attention several months after submitting this paper, contains a different proof of basically the same theorem as Theorem 2.)

IV. Illustrative Examples

Example 1 (Theorem 1): Consider the Petri net N shown in Fig. 1. Note that N has all transitions with outdegree ≤ 1. The incidence matrix A of this net and an integer solution $X \geq 0$ for the homogeneous equation, $A^T X = 0$ is shown below. Note that X is a T-invariant such that $X \geq 0$ and $X(8) \neq 0$, where T_8 is the goal transition.

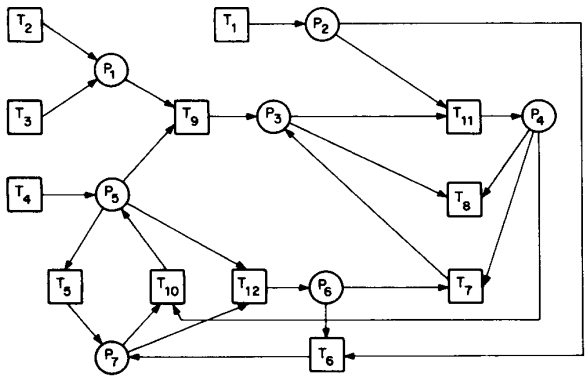


Fig. 1. The Petri net N for Example 1 illustrating Theorem 1.

$$
\begin{array}{c}
\\ P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \\ P_7
\end{array}
\begin{array}{c}
\begin{array}{cccccccccccc} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 & T_{10} & T_{11} & T_{12} \end{array} \\
\begin{bmatrix}
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & -1 & 1 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & -1 & 0 & -1
\end{bmatrix}
\end{array}
\begin{bmatrix} 2 \\ 2 \\ 0 \\ 3 \\ 0 \\ 1 \\ 0 \\ 1 \\ 2 \\ 0 \\ 1 \\ 1 \end{bmatrix}
=
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
$$

Therefore, all conditions required by Theorem 1 are satisfied. Theorem 1 implies that there exists another T-invariant $X' \geq 0$, $X'(8) \neq 0$ which would be executable from $M_0 = 0$.

The T-invariant X is not executable from $M_0 = 0$. This can be easily verified because $X(12) \neq 0$. T_{12} cannot be fired without a token in P_7. A token can only be placed into P_7 by firing T_6 (T_5 cannot be fired because $X(5) = 0$). T_6, however, cannot fire until T_{12} fires and deposits a token in P_6. Thus, there is a deadlock condition and X cannot be executed from $M_0 = 0$.

Let us follow the proof of Theorem 1 in an attempt to find X' and the corresponding firing sequence σ'''. $B = \{T_1, T_1, T_2, T_2, T_4, T_4, T_4, T_6, T_8, T_9, T_9, T_{11}, T_{12}\}$. Construct the firing sequence σ' to satisfy the four conditions stated in the proof of Theorem 1. This is very easy to do. Simply rearrange as many of the transitions from B into any firing sequence which would be firable from $M_0 = 0$ as is possible. Theorem 1 guarantees that T_8 will be contained in any such sequence. $\sigma' = < T_1, T_1, T_2, T_2, T_4, T_4, T_4, T_9, T_9, T_{11}, T_8 >$.

When all the transitions in σ' are fired from $M_0 = 0$, it produces the marking $M = [0\ 1\ 0\ 0\ 1\ 0\ 0]^T$. σ''' can be constructed by removing the last occurrence of T_1 and the last occurrence of T_4 from σ', that is, $\sigma''' = < T_1, T_2, T_2, T_4, T_4, T_9, T_9, T_{11}, T_8 >$. σ''' now reproduces the empty marking. $\bar{\sigma}''' = X' = [1\ 2\ 0\ 2\ 0\ 0\ 0\ 1\ 2\ 0\ 1\ 0]^T$. It can easily be verified that $A^T X' = 0$. X' is therefore a T-invariant and $\| X' \| \subseteq \| X \|$.

Example 2 (Propositional Logic):

Given the following set of clauses

1) A
2) B
3) $A \wedge B \Rightarrow C$
4) $B \wedge C \Rightarrow D$
5) $D \Rightarrow A$
6) $D \Rightarrow C$

we wish to prove that $D \wedge C$.

So we add the negation of $D \wedge C$ to the set of clauses and obtain the following set which is written in conjunctive normal form.

1) A
2) B
3) $C \vee \neg A \vee \neg B$
4) $D \vee \neg B \vee \neg C$
5) $A \vee \neg D$
6) $C \vee \neg D$
7) $\neg D \vee \neg C$

We will now try to prove that this set of clauses contains a contradiction.

The clauses are in Horn clause form. So we will apply Theorem 2 to test for the existence of a contradiction. Fig. 2 shows the net representation for this set of clauses.

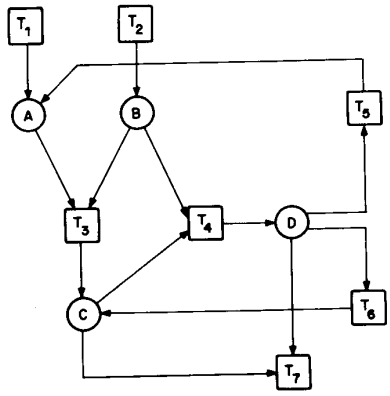


Fig. 2. Petri net model of the set of clauses for Example 2 illustrating Theorem 2.

Below is the incidence matrix A for the net N and an integer solution to the homogeneous equation $A^T X = 0$. Note that the incidence matrix can be obtained directly from the set of clauses according to the following rule. If a literal appears more than once in a clause, the clause can be simplified until every literal appears at most once. $a_{ij} = 1, -1$, or 0 depending on whether literal j appears in clause i in nonnegated form, negated form, or not at all, respectively.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{ccccccc} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 \end{array} \\ \begin{bmatrix} 1 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 & -1 & -1 & -1 \end{bmatrix} \end{array} \begin{bmatrix} 2 \\ 3 \\ 2 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The above X is a T-invariant such that $X \geq 0$ and $X(7) \neq 0$ (T_7 is the goal). Therefore all conditions required by Theorem 2 are met and the set of clauses contains a contradiction. Consequently, $D \wedge C$ is proved to be true.

Remarks: This example illustrates how Theorem 2 transforms a propositional logic problem into a linear algebra problem. Specifically, in propositional logic, proof by contradiction (refutation) can be accomplished by purely linear algebraic methods, or structural properties of the net representation.

Example 3 (Predicate Logic): Consider the following logic program, which is expressed in a notation similar to PROLOG where each clause is interpreted as, for example, $A :- B, C, D$ mean $B \wedge C \wedge D \Rightarrow A$.

1) $P(x, y) :- F(x, y)$
2) $P(x, y) :- M(x, y)$
3) $G(x, y) :- P(x, z), P(z, y)$
4) $Q(x, y) :- M(x, z), F(y, z)$
5) $M(x, y) :- Q(x, z), F(z, y)$
6) $F(x, y) :- Q(x, z), M(z, y)$
7) $Q(x, y) :- Q(y, x)$
8.1) F(Bob, Bill) :-
8.2) F(Bob, Amy) :-
8.3) F(Bill, John) :-
8.4) F(Bill, Jack) :-
8.5) F(Bill, Paul) :-
8.6) F(Jack, Larry) :-
8.7) F(Jack, Steve) :-
8.8) F(Andy, Janet) :-
8.9) F(Andy, Randy) :-
8.10) F(Randy, Tim) :-
8.11) F(Randy, Tom) :-
8.12) F(Tom, Linda) :-
8.13) F(Fred, Mary) :-
9.1) $:- G(x,$ Steve)

Clauses 1-7 are procedure declarations. They involve five predicates which are listed below:

$F(x, y)$ means that x is the father of y
$M(x, y)$ means that x is the mother of y
$P(x, y)$ means that x is the parent of y
$G(x, y)$ means that x is the grandparent of y
$Q(x, y)$ means that x is married to y.

The program is a simplified model of normal family relationships.

Clauses 8.1–8.13 are facts. The numbering was chosen to imply that these are actually only different colorings of the same transition. For brevity of a high-level net representation of the program, facts 8.1–8.13 will be replaced by a single transition denoted by

8) $F(x, y) :-$

whose set of colors $\{ < x, y > \}$ is the set of 13 pairs of names in the arguments of clauses 8.1–8.13. This method of representing a logic program by a high-level net is different from that presented in [3] where the 13 source transitions are drawn corresponding to the 13 facts.

Clause 9 is the goal clause. The x in clause 9 represents an unknown variable. The purpose of the program is to find substitution(s) for x and thus determine who is(are) Steve's grandparent(s).

The variables x, y, z are used in the program. The scope of each variable is the clause within which it occurs. When a variable is instantiated, assigned a value, the same value must be assigned to all instances of that variable within the clause. A variable in a different clause, even though

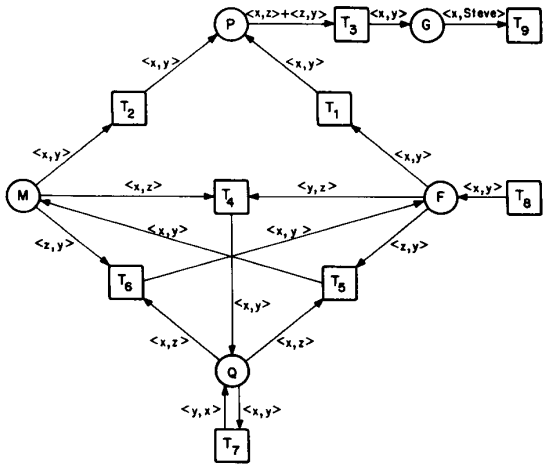


Fig. 3. High-level Petri net model of the logic program for Example 3.

it has the same name, is a different variable and can be assigned a different value.

The high-level Petri net model of the above program can be drawn by a procedure similar to the one given in [3] and is shown in Fig. 3.

The incidence matrix A for this net is given by

$$A = \begin{array}{c} \\ T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \\ T_9 \end{array} \begin{array}{c} \begin{array}{cccccc} \text{---}M\text{---} & & \text{---}F\text{---} & & \text{----}P\text{----} & \end{array} \\ \left[\begin{array}{cccccc} 0 & & -\langle x, y\rangle & & \langle x, y\rangle & \\ -\langle x, y\rangle & & 0 & & \langle x, y\rangle & \\ 0 & & 0 & & -\langle x, z\rangle - \langle z, y\rangle & \\ -\langle x, z\rangle & & -\langle y, z\rangle & & 0 & \\ \langle x, y\rangle & & -\langle z, y\rangle & & 0 & \\ -\langle z, y\rangle & & \langle x, y\rangle & & 0 & \\ 0 & & 0 & & 0 & \\ 0 & & \langle x, y\rangle & & 0 & \\ 0 & & 0 & & 0 & \end{array}\right. \end{array} \begin{array}{c} \begin{array}{cc} \text{----}G\text{----} & \text{----}Q\text{----} \end{array} \\ \left[\begin{array}{cc} 0 & 0 \\ 0 & 0 \\ \langle x, y\rangle & 0 \\ 0 & \langle x, y\rangle \\ 0 & -\langle x, z\rangle \\ 0 & -\langle x, z\rangle \\ 0 & +\langle x, y\rangle - \langle y, x\rangle \\ 0 & 0 \\ -\langle x, \text{Steve}\rangle & 0 \end{array}\right] \end{array}$$

In general, the incidence matrix A can be obtained (by inspection or automatically on a computer) from a logic program by the following procedure:

Step 1: Each clause in the program will be one row of the incidence matrix.

Step 2: Each distinct predicate in the program will be one column of the incidence matrix.

Step 3: In the logic program, prefix all entries to the right of the :− with a minus sign. These entries are considered to be negative. The entries on the left of the :− are considered positive.

Step 4: Eliminate the :− from the program.

Step 5: Rearrange each line of the program so that the predicates are in the proper column of the matrix.

Step 6: Remove the predicate names and leave only the arguments enclosed in < >.

Step 7: If any element of the matrix has more than one entry, express them as a formal sum to form a single entry.

The next step is to find a T-invariant X for the net shown in Fig. 3. There have been proposed several methods for finding T-invariants of high-level Petri nets [7], [9]–[12]. To describe these is beyond the scope of this paper. In what follows, we use the example to illustrate some of the procedures involved in finding T-invariants.

For the high-level Petri net shown in Fig. 3, it is verified below that there exists a T-invariant X such that $A^T X = 0$, $X \geq 0$, $X(9) \neq \varnothing$. For example, consider the following vector of substitutions.

$$X = \begin{matrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \\ T_9 \end{matrix} \begin{bmatrix} \{x = a, y = c\} \cup \{x = c, y = b\} \\ \varnothing \\ \{x = a, y = b, z = c\} \\ \{x = d, y = e, z = f\} \\ \{x = d, y = f, z = e\} \\ \varnothing \\ \varnothing \\ 2\{x = e, y = f\} \cup \{x = a, y = c\} \cup \{x = c, y = b\} \\ \{x = a, y = b\} \end{bmatrix}$$

The notation in which the above vector X (T-invariant) is expressed is interpreted as follows. Transition T_1 fires 2 times. During the first firing, the values a and c are assigned to the variables x and y for all the arcs incident on T_1. During the second firing, the values c and b are assigned to x and y. We have used the term first firing and second firing but the order in which these transitions occur in a firing sequence is not known. The order in which the firings are listed in the T-invariant is arbitrary. The other transitions are interpreted analogously and the symbol $\varnothing$ indicates that the transition does not fire at all. T-invariants of a high-level Petri net are vectors of substitutions and sometimes are referred to as *object T-invariants* [11].

The multiplication of A^TX is accomplished by substituting the values for the variables according to the substitution rules stated in the T-invariant and adding all the terms in each row. For our example, the multiplication of A^TX yields a zero vector as is illustrated below.

$$\begin{matrix} M \\ F \\ P \\ G \\ Q \end{matrix} \begin{bmatrix} -<d,f> + <d,f> \\ -<a,c> - <c,b> - <e,f> - <e,f> + 2<e,f> + <a,c> + <c,b> \\ <a,c> + <c,b> - <a,c> - <c,b> \\ <a,b> - <a,b> \\ <d,e> - <d,e> \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The scope of each variable in the logic program is the entire clause in which it occurs. A clause corresponds to a row in A or a column in A^T as was shown previously. In multiplying A^TX each row of A is "multiplied" by a single element of X, where this "multiplication" corresponds to a substitution of values. Therefore, the same substitutions will be made for all entries in a row. Thus, the scope rules are preserved in the multiplication because the same substitutions are made for all occurrences of a variable within a clause.

The values $\{a, b, c, d, e, f\}$ which were substituted for the variables $\{x, y, z\}$ are themselves variables but from a different sort. They are variables from the T-invariant and their scope is the entire program. Thus, their scope extends over the entire matrix A and the terms in the multiplication of A^TX can be added. This produces a column vector which is all zeros. Thus $A^TX = 0$ and X is a T-invariant. It is important to understand the scope of each variable and ensure that it is not violated during any operation.

The T-invariant contains variables which must now be instantiated (assigned values) from the set of colors. The set of colors is defined for transitions T_8 and T_9 as shown earlier in the logic program. The set of colors for the other transitions is not defined. It would appear that the set of colors for the other transitions is infinite but that is not the case. The set of colors for the other transitions is finite and can be found from unfolding the net into an uncolored net and identifying which colors are possible in the other transitions. Fortunately, the set of colors for the other transitions does not need to be determined. All that is required is to instantiate the variables in X.

The variables are instantiated by choosing appropriate colorings for all transitions which have a set of colors in an attempt to satisfy the requirements of X. This is shown below:

from 9.1 and T_9 b = "Steve"
from 8.7 and T_8 c = "Jack"
from 8.4 and T_8 a = "Bill"

The variables $\{d, e, f\}$ cannot be uniquely identified. From T_8, $\{e, f\}$ can assume any coloring from 8.1 to

8.13. Let e = "Bob" and f = "Bill". The variable d is totally unrestricted and can be assigned any value at all. Let d = "Wilma".

We now have a fully instantiated T-invariant X which is shown below.

$$X = \begin{matrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \\ T_9 \end{matrix} \begin{bmatrix} \{x = \text{Bill}, y = \text{Jack}\} \cup \{x = \text{Jack}, y = \text{Steve}\} \\ \varnothing \\ \{x = \text{Bill}, y = \text{Steve}, z = \text{Jack}\} \\ \{x = \text{Wilma}, y = \text{Bob}, z = \text{Bill}\} \\ \{x = \text{Wilma}, y = \text{Bill}, z = \text{Bob}\} \\ \varnothing \\ \varnothing \\ 2\{x = \text{Bob}, y = \text{Bill}\} \cup \{x = \text{Bill}, y = \text{Jack}\} \cup \{x = \text{Jack}, y = \text{Steve}\} \\ \{x = \text{Bill}, y = \text{Steve}\} \end{bmatrix}$$

All conditions required by Theorem 2 are satisfied. The problem is solved and Bill is Steve's grandparent.

Suppose that in addition to providing the answer "Bill" the computer is asked to show the line of reasoning which proves that Bill is Steve's grandparent. For this, a firing sequence needs to be found which will correspond to the T-invariant X and be executable from the empty marking. No such firing sequence exists as can be easily verified by attempting to find one. Theorem 1 or Corollary 3 states that there exists a T-invariant X' and a corresponding firing sequence σ' which is firable from the empty marking and which fires the goal transition. The proof of Theorem 1 gives a procedure by which this firing sequence can be easily constructed. This construction is illustrated in Example 1. Only the answer is presented here.

The firing sequence can be expressed as a sequence of instantiated clauses from the original logic program. This is shown below.

$F(\text{Bill}, \text{Jack}) :-$
$F(\text{Jack}, \text{Steve}) :-$
$P(\text{Bill}, \text{Jack}) :- F(\text{Bill}, \text{Jack})$
$P(\text{Jack}, \text{Steve}) :- F(\text{Jack}, \text{Steve})$
$G(\text{Bill}, \text{Steve}) :- P(\text{Bill}, \text{Jack}), P(\text{Jack}, \text{Steve})$
$:- G(\text{Bill}, \text{Steve})$

Resolution can now be used to show that this set of clauses contains a contradiction and thus prove that Bill is Steve's grandparent.

V. Conclusion

In summary, the present high-level Petri net method has converted the processes involved in logic programming

$$X' = \begin{matrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \\ T_9 \end{matrix} \begin{bmatrix} \{x = \text{Bill}, y = \text{Jack}\} \cup \{x = \text{Jack}, y = \text{Steve}\} \\ \varnothing \\ \{x = \text{Bill}, y = \text{Steve}, z = \text{Jack}\} \\ \varnothing \\ \varnothing \\ \varnothing \\ \varnothing \\ 2\{x = \text{Bill}, y = \text{Jack}\} \cup \{x = \text{Jack}, y = \text{Steve}\} \\ \{x = \text{Bill}, y = \text{Steve}\} \end{bmatrix}$$

$$\sigma' = < T_8\{x = \text{Bill}, y = \text{Jack}\}, T_8\{x = \text{Jack}, y = \text{Steve}\},$$
$$T_1\{x = \text{Bill}, y = \text{Jack}\}, T_1\{x = \text{Jack}, y = \text{Steve}\},$$
$$T_3\{x = \text{Bill}, y = \text{Steve}, z = \text{Jack}\}, T_9\{x = \text{Bill}, y = \text{Steve}\} >$$

such as binding, unification, substitution, and resolution into a problem of solving a system of homogeneous equations for a particular kind of solutions called T-invariants. The main motivation of this work is to provide new insights and control strategies for parallel processing of logic programs and automatic theorem proving.

From a practical standpoint, the key step is obviously finding the T-invariant for the high-level net. This is a topic of current research which has already produced some practical results [7], [9]–[12], which can be used to find a T-invariant. However, a general method and computational cost of finding T-invariants applicable to logic programming are unknown and suggested for further study.

References

[1] J. W. Lloyd, *Foundations of Logic Programming*. Berlin: Springer-Verlag, 1984.

[2] N. J. Nilsson, *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga, 1980.

[3] T. Murata and D. Zhang, "A predicate-transition net model for parallel interpretation of logic programs," *IEEE Trans. on Software Eng.*, vol. 14, no. 4, pp. 481–497, Apr. 1988; also, an earlier version, "A high-level Petri net model for parallel interpretation of logic programs," in *Proc. 1986 Int. Conf. Computer Languages*, IEEE Comput. Soc., Oct. 27–30, 1986, pp. 123–132.

[4] K. Lautenbach, "On logical and linear dependencies," Sankt Augustin, Germany, GMD Rep. 147, 1985.

[5] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[6] T. Murata, "Modeling and analysis of concurrent systems," in *Handbook of Software Engineering*, C. R. Vick and C. V. Ramamoorthy, Eds. New York: Van Nostrand Reinhold, 1984, pp. 39–63.

[7] K. Jensen, "Coloured Petri nets and the invariant-method," *Theoretical Comput. Sci.*, vol. 14, pp. 317–336, 1981.

[8] J. L. Peterson, "A note on high-level Petri nets," *Inform. Processing Lett.*, vol. 11, no. 1, pp. 40–43, Aug. 1980.

[9] H. J. Genrich and K. Lautenbach, "System modeling with high-level Petri nets," *Theoretical Comput. Sci.*, vol. 13, pp. 109–136, 1981.

[10] K. Jensen, "How to find invariants for coloured Petri nets," in *Mathematical Foundations of Computer Science (Lecture Notes in Computer Science*, vol. 118), 1981, pp. 327–338.

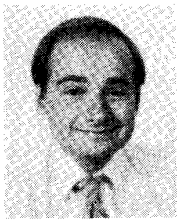
[11] K. Lautenbach and A. Pagnoni, "Invariance and duality in predicate-transition nets and coloured nets," Sankt Augustin, Germany, GMD Rep. 132, 1985.

[12] M. Silva *et al.*, "Generalized inverses and the calculation of symbolic invariants for coloured Petri nets," *Technique et Saience Informatiques*, vol. 4, no. 1, pp. 113–126, 1985.

[13] G. Peterka, "Colored Petri net T-invariant method for logic programs," Master's thesis, Univ. Illinois at Chicago, 1986.

[14] A. Sinachopoulos, "Derivation of a contradiction by resolution using Petri nets," *Petri Net Newslett.*, vol. 26, pp. 16–29, Apr. 1987.

[15] D. Zhang and T. Murata, "Fixpoint semantics for predicate-transition net model for Horn clause logic programs," Univ. Illinois at Chicago, Tech. Rep. EECS-87-2, Apr. 1987; also to appear in *Advances in Theory of Computation and Computational Mathematics*, vol. 1. Norwood, NJ: Ablex, 1989.



George Peterka was born on January 29, 1960 in Uhersky Brod, Czechoslovakia, and emigrated to the United States in 1968. He received the B.S. degree in electronic engineering and the M.S. degree in computer science from the University of Illinois at Chicago in 1981 and 1986, respectively.

From 1981 through 1985 he held the position of Development Engineer at AT&T Technologies, Network Systems. He left AT&T in 1985 and returned to the University of Illinois at Chicago to work on the M.S. degree. He is currently employed as a Development Engineer for Illinois Advanced Design, Inc., Hillside, IL.

Tadao Murata (S'62–M'66–SM'77–F'85) received the M.S. and Ph.D. degrees in Electrical Engineering from the University of Illinois at Urbana in 1964 and 1966, respectively.

He is presently a Professor of Electrical Engineering and Computer Science at the University of Illinois at Chicago. During occasional leaves of absence from the University of Illinois, he taught at the University of California at Berkeley and Tokai University, Tokyo, Japan, and was invited to visit Petri's Institute of GMD mbH in Germany and several other research institutes and universities in Europe. His current research interests include applications and theory of Petri nets, concurrent computer systems, and data flow and parallel computations. In these areas, he has published extensively and been awarded several National Science Foundation research grants since 1976. Prior to that, he worked in the area of circuits, systems, and applied graph theory. He has served on the U.S. National Academy of Sciences/Computer Science and Technology Board panels.

He is an editor for the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING and served as the general chairman for the 1987 International Workshop of Petri Nets and Performance Models. He is a member of the Association for Computing Machinery, EATCS, IECE, and the Information Processing Society of Japan. He is listed in *Who's Who in Engineering* and *Who's Who in America*.