Non-deterministic Modelling of Logical Queries in Deductive Databases

Hussien Aly* and Z Meral Ozsoyoğlu†

Computer Engineering and Science Department
Case Western Reserve University
Cleveland, Ohio 44106

ABSTRACT

We propose a technique based on Petri Nets formalism to model logic queries in deductive databases The model is called PNLP (Petri Net model for Logic Programs), and it has a simple formal description and a graphical representation The PNLP model explicitly represents the relationships between rules and predicates It is general and flexible enough to demonstrate the flow of control in different algorithms used to evaluate recursive logic queries In fact the model unifies the level of description of these algorithms, and facilitates identifying similarities and differences between them The inherent non-determinism in the PNLP model may also be useful in recognizing the parallelism within Horn-clause logic programs In this paper, the PNLP model is described, and its functionality is demonstrated by modeling several existing algorithms for recursive query evaluation

1. Introduction

A deductive database consists of a set of base relations called *extensional database* (EDB), and a set of function-free Horn clauses which define virtual relations, called *intensional database* (IDB) These clauses are also called *rule* A general presentation of the subject can be found in [GM]

The problem of evaluating logical queries in deductive databases especially the evaluation of recursive queries, has attracted the attention of several researchers Various approaches to evaluate recursive queries appear in the literature, e g [BR],[BMSU],[U],[HN],[SZ] However most of these algorithms are not clearly presented, are hard to understand, and they are presented in different levels of details [BR] Many of them look very similar except for few details Very little has been done to compare these algorithms, or to identify the impact of the differences in details on the behavior and performance of these algorithms [BR] We believe that this is due to the lack of a simple model that can capture the interrelationships between predicates and rules and have the ability to demonstrate the way constants are propagated through various applications of the rules Such a model should also facilitate recognizing the flow of control of various algorithms that are used for evaluating the queries, and eventually should help in choosing the best strategy

In this paper, we propose a non-deterministic model based on Petri Nets (PN) formulation [P], called the *Petri Net model for Logic Programs* (PNLP) We modify the definition of PN, and allow the elements to have some information (colors) [P, A] so that they will be able to simulate basic operations in the resolution procedure [K], and the relational algebra The PNLP is non deterministic in the sense that given an initial state, there are, in general

* Research is supported in part by a scholarship from Egyptian government

† Research is supported in part by the NSF under grant No DCR 86-05554

more than one operation leading to different states This gives the model a natural way of recognizing parallelism in logic programs, and the necessary synchronization

The PNLP is motivated by a research effort for developing an automata for evaluating logic queries with the ability to choose an optimum flow of control This automata should use both the static characteristics and the existing knowledge about the database, and the dynamic facts discovered during execution We first present the PNLP model and its characteristics Then we model several existing recursive query processing algorithms in the PNLP to illustrate how the PNLP unifies the level of presentation of the various algorithms in terms of a relatively simple language

The rest of the paper is organized as follows Section 2 gives the necessary background for Petri Nets and their terminology In section 3, the PNLP model is presented, and its structure aspect and the dynamic aspect are discussed In section 4, we discuss some related works In section 5, some well known algorithms are presented as examples to demonstrate their flow of control in terms of the PNLP language In section 6, a general discussion of these algorithms and their PNLP models are given The last section is the conclusion and future work

2. Petri Nets - Background

Petri Nets (PN) is a very powerful modelling technique that has both static and dynamic aspects In the literature, PNs are usually mentioned when there are parallel asynchronous problems A detailed presentation of the theory and the modelling power of PN can be found in [P]

A PN *structure* is a quadruple (P, T, I, O), where P and T are finite sets whose elements are called *places* and *transitions* respectively, I is the *input function* from T into bags of places (I T $\rightarrow$ P^{∞}), and O is the *output function* from T to bags of places (O T $\rightarrow$ P^{∞}) A place $p \in P$ is called an *input place* to transition t if it occurs in the input bag of this transition, i e #(p, I(t)) > 0 (# is the number of occurences function in Bag theory, and it is analogous to the membership function in set theory) Similarly, p is an *output place* of the transition t if #(p, O(t)) > 0

The PN structure also has a graphical representation Places and transition are represented as circles ○ and bars | respectively Input and output functions are represented by arcs from (to) places to (from) transitions respectively

Example 2.1

$P = \{p_1, p_2, p_3, p_4\}$, $T = \{t_1, t_2, t_3\}$
$I(t_1) = Bag\{p_2\}$, $O(t_1) = Bag\{p_1\}$,
$I(t_2) = Bag\{p_4\}$, $O(t_2) = Bag\{p_2, p_2, p_3\}$,
$I(t_3) = Bag\{p_3\}$, $O(t_3) = Bag\{p_4\}$

The graphical representation of the above structure is given in Figure 2 1

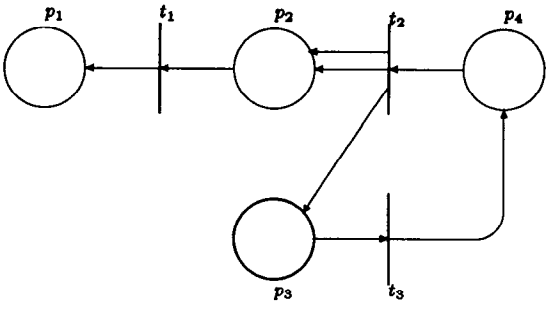

Figure 2 1 Graphical representation for example 2 1

A *marked PN* is defined as a PN structure and an assignment of tokens to the places A *token* is a primitive concept for PN, and is represented graphically by '•' The *marking function* defined as μ P $\rightarrow$ N, where N is the set of natural numbers, and $\mu(p)$ is the number of tokens in a place p The *state* of a place p is its marking, i e $\mu(p)$, and the *state of the net* is defined by the markings of all the places The *state space* of a PN is the set of all possible markings of the net

A PN executes (i e changes its state) by *firing* the transitions A transition t can be fired if it is *enabled*, i e all its input places have sufficient tokens Formally, t is enabled if $\mu(p) \geq \#(p, I(t))$ for all input places p for this transition When fired, the transition t removes the tokens from input places and produces new tokens into output places (theoretically in no time)

There are many extensions to the original PN discussed above These extensions try to improve and increase the modelling power of these nets by adding exclusive-or and inhibitor arcs to the model Also, some extensions provide some information to be attached to the net elements (coloring for the net), and time for the transition firing [A] Unfortunately the basic analysis techniques for PN are not applicable to the most of these extensions, especially to the colored PN [CJ] Notice that our PNLP model is in principle a colored extension of Petri Nets Another approach for studying PN is by considering them as formal language automata The sequence of transitions that are fired are viewed as strings, and the language accepted by the net is the set of all possible firing sequences of the transitions So, Petri Nets behavior may be studied by studying the properties of their languages

3. The Model

The PNLP model has two aspects The *structure aspect* and the *execution aspect* These are also called static and dynamic aspects respectively

3.1. Structure aspect

The *PNLP structure* displays the interrelationships between rules and predicates as specified by the logic program We will first present the formal definition, and then discuss the graphical representation

Definition 3.1 The structure of a PNLP model is a quadruple (T,P,I,O), where
T is a non-empty set of transitions,
P is a non-empty set of places, and $P \cap T = \emptyset$,
I $T \rightarrow P^{\infty}$ is the input function from T to bags of places,
O $T \rightarrow P$ is the output function from T to P
A transition $t \in T$ in PNLP is a tuple of the form (c, η, ξ) where c is an integer called the *count*, η is a list of *output arguments*, and ξ is a set of *constraints* of the form $x\ \theta\ y$, $\theta \in \{=, >, <, \geq, \leq\}$

A logic program consists of function-free Horn rules $r_1, r_2, \ldots, r_m$, and each rule r is of the form

$$p(x_i, \ldots, x_j) \text{ - } p_l(x_q, \ldots, x_s), \ldots, p_k(x_u, \ldots, x_v)$$

We can set up the following isomorphism, M, between such a logic program and a PNLP structure Informally, we represent each rule by a transition, and each predicate by a place Predicates that occur in the rule body are input places to the transition representing this rule and the predicate in the head will be the output place More formally, M is defined as follows

M(T) = { r | r is a rule in the logic program },
M(P) = { p | p is a predicate name in the program },
M(I(t))= Bag { p | p $\in$ M(P) and p appears in the body of M(t) },
M(O(t)) = p , p $\in$ M(P) and p is the head of M(t), and if M(t) = r then
η = list of variables in the head of rule r, and
ξ = specifies the relationship between the variables in the body of the rule r

Example 3.1 Suppose we have the following logic program [HN]

```
r1   s ( X1,Z1 ) - m ( X1,Y1 ), t ( Y1,Z1 )
r2   t ( Y1,Z1 ) - p ( W1,Z1 ), s ( Y1,W1 )
r3   t ( Y1,Z1 ) - f ( Y1,Z1 )
```

m, f, and p are base relations, and can be interpreted as mother, father, and parent respectively s can be thought as a particular kind of ancestor, and t is an auxiliary relation used to define s The PNLP structure corresponding to the above program will be as follows *

* We use the notation p[i] to denote the i^{th} argument in the predicate p

$T = \{ r_1, r_2, r_3 \}$, $P = \{ f, m, p, s, t \}$,
$r_1 = (c_1, [m[1], t[2]], \{ m[2]=t[1] \})$,
$r_2 = (c_2, [s[1], p[2]], \{ p[1]=s[2] \})$,
$r_3 = (c_3, [f[1], f[2]], \emptyset)$,
$I(r_1) = Bag\{ m,t \}$, $O(r_1) = s$,
$I(r_2) = Bag\{ p,s \}$, $O(r_2) = t$,
$I(r_3) = Bag\{ f \}, O(r_3) = t$

Graphical representation A nice property of PNLP is that it has a graphical representation of the formal description in a relatively simple graph A *PNLP graph*, in general, is a labeled bipartite-directed multigraph Places are represented by circles ○, and transitions by bars | Input function is simulated by arcs from places to transitions, while the output function is represented by arcs from transitions to places The list of output variables and the set of constraints of a transition are represented as labels on the output and input arcs The graphical representation of the program in example 3 1 is given in Figure 3 1

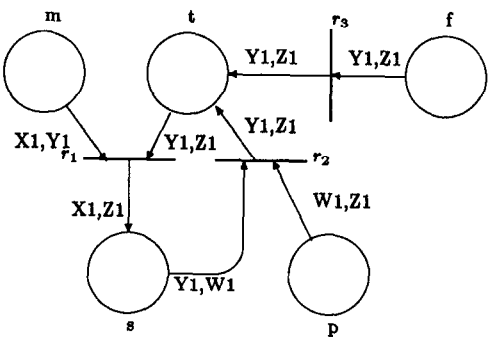

Figure 3 1 Graphical representation for example 3 1

Example 3.2 Consider the following definition of the transitive closure tr of a relation e(X, Y)

r_1 tr(X, Y) - e(X, Y)
r_2 tr(X, Y) - e(X, Z), tr(Z, Y)

The graphical representation is shown in Figure 3 2

3.2. Marked net and the state space

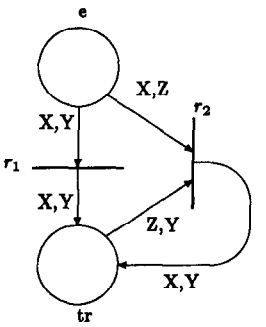

Figure 3 2 The transitive closure example

The net is said to be *marked* if we allow tokens to be assigned to places A *token* is a primitive concept in PN based models In PNLP, a token represents a set of tuples that satisfy the relation represented by the place So, in this sense, tokens are colored (i e contain information) in PNLP A token may be of one of the three types below

- *Tuple token* The token represents a set of fully instantiated tuples
- *Query token* Represents a set of partially instantiated tuples
- *Free token* represents a set of totally un-instantiated tuples

Un-instantiated attributes in a token will be represented by '?'

Example 3.3 Suppose, in example 3 2, we have the query tr(a, ?) This will be represented as a query token (a, ?) in tr place, and graphically as in Figure 3 3

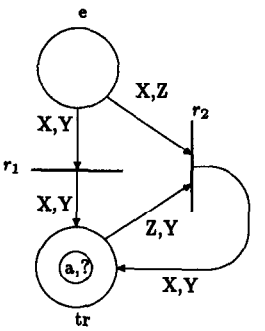

Figure 3 3 Representation of the query token tr(a,?)

Let $P = \{p_1, p_2, \ldots, p_m\}$ be the set of places in a marked PNLP, and τ_i denote the current set of tokens assigned to the place p_i Then τ_i is the state of p_i, and the *state* of a marked PNLP is the ordered tuple $(\tau_1, \tau_2, \ldots, \tau_m)$ where $m = |P|$ The *state space* S is the set of all possible states,

$$S = P(D_1) \times P(D_2) \times \cdots \times P(D_m)$$

where D_i denotes all possible instances of the relation p_i with the symbol '?' allowed,† and $P(D_i)$ is the power set of D_i The *reachability* set from an initial state of the PNLP is the set of all possible states (markings) that can be reached by executing the net

3.3. Execution aspect

The *marking function* μ is a mapping from the set of places to sets of tokens $\mu(p)$ will give the current state of the place p, i e the set of tokens assigned to p In the original non-colered Petri Nets, the net executes only by firing the transitions [P] Since the PNLP is a colored extension of Petri Nets, we need to modify the execution rules, and define some new concepts concerning the execution of the PNLP The idea behind these modifications is that we need to imitate the behavior of the resolution procedure [K] The execution rules of PNLP are given below

Fire rule (FIR) A transition $t \in T$ is said to be *enabled* if there are some tokens in all its input places, i e $\mu(p) \neq \emptyset$, for all $p \in I(t)$ A transition $t = (c, \eta, \xi)$ may *fire* if it is enabled Firing a transition, in general, will change the state of the *output place* Informally, FIR will project into the output place, the result of the join operation* on the input tokens Let μ and μ' be the marking before and after firing t respectively Then for an output place $p = O(t)$ we have

$$\mu'(p) = \mu(p) \cup \pi_\eta (\rhd\lhd_\xi v_i)$$

where $v_i \in \mu(p_i)$ and $p_i \in I(t)$

Above, π_η is the projection over the attributes in η, and $\rhd\lhd_\xi$ means the join operation according to the constraints in ξ The state of the *input places* may also change as follows

For all $p \in I(t)$,
$\mu'(p) = \mu(p)$ - { v | v $\in \mu(p)$ and v has a non-empty contribution to the result of the join above}

If the firing changes the state of the input places, we say that firing *consumes* input tokens This is valid when the consumed tokens have no further non-redundant contribution

Perturbation rule (PRT) A transition $t \in T$ is a *candidate for perturbation* if there exist a query token in the output place of this transition

A transition $t = (c,\eta, \xi)$ may be *perturbed* (PRT) if it is candidate for perturbation (Recall that η is a list of output variables and ξ is a set of constraints) Intutionally, we can think of PRT as the process of propagating the constants in the query to the rule body and generating a request for evaluation of each predicate in the rule body In general, perturbing the transition will change the state of the *input places* as follows

Let μ and μ' be the marking before and after perturbation of t Let v be the query token which make the transition candidate for perturbation Let us define the operation $\nabla(p)$ of creating a new token into a place p as follows
$\nabla(p)$ = a new token of type free, if the intersection of η and the argument of p is empty,
= a query token with common arguments instantiated from v, otherwise

Now, PRT will change the states of the input places so that

† Note that, in general, a token represents a subrelation

* This is usually a natural join

$$\mu'(p) = \mu(p) \cup \nabla(p), \quad p \in I(t)$$

The state of the output place may also change due to PRT The new marking will be

$$\mu'(p) = \mu(p) - \{ v \}, \quad p \in O(t)$$

In this case we say that the perturbation *consumes* the input query token
For instance, in example 3 3, perturbing r_2 will change the state of the net as shown in Figure 3 4 If PRT consumes the input query token, then the token (a,?) in tr will be removed

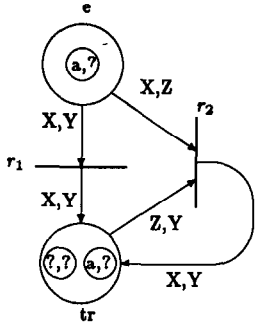

Figure 3 4 The state of the net after perturbation of r_2

For some optimization purposes, as we will see in section 5, the transition may count the number of PRT it makes (in the count attribute), and produce an equivalent number of free tokens into the appropriate places Also the transition will be able to mark the output tokens using the current value of the count

Information passing rule (PAS) A transition $t = (c, \eta, \xi)$ may *pass information* (PAS) from some token $v_1 \in \mu(p_1)$ into another token $v_2 \in \mu(p_2)$ where $p_1, p_2 \in I(t)$ in two ways

a- *Sideway information passing* If there exist an equality constraint $p_1[i] = p_2[j] \in \xi$ and $p_1[i]$ is instantiated but $p_2[j]$ is not The result is the instantiation of $v_2[j]$

b- *Cycle-stealing information passing* * If there exist an equality constraint $p[k] = p_2[j] \in \xi$, and $p_1[i] \in \eta$ such that $p_1[i]$ will be the k^{th} argument of p if the transition is fired, and $p_1[i]$ is instantiated but $p_2[j]$ is not

* The name cycle-stealing is because this is equivalent to firing the transition then making sideway information passing, without actually firing the transition

Example 3.4 Assume that in example 3 3 we have the following marking

$$\mu(e) = \{(a, b), (?, ?)\},$$
$$\mu(tr) = \{(a, ?)\}$$

Now transition r_2 can pass information from the token (a,b) to the token (?,?) (in the same place), since we have the constraint $e[1] = tr[2]$, and $e[2]$ is in the output list η Then $e[2]$ will be the 2^{nd} argument of the token, if r_2 happens to fire So, we can use $e[2]$ to bind the 1^{st} argument in the other token The new markings will be

$$\mu(e) = \{ (a, b), (b, ?) \},$$
$$\mu(tr) = \{ (a, ?) \}$$

3.4. The PNLP language

Another way for studying PNLP is to focus not on what markings (states) are reachable, but on how they are reached Of particular interest are the sequences of operations (FIR, PRT, PAS) which lead from one state to another Such a sequence of operations is called a string, and a set of strings is a language We will call this *the language defined by PNLP*, or simply the *PNLP language* This is similar to the definition of Petri Net languages [P] A complete formulation of this language is not given in this paper Here, we are only interested in using the language symbols to describe the flow of control in the algorithms in a clear and concise manner The relevant symbols and their associated meanings are given in table 3 4-1

Below, we discuss some properties of the PNLP language with respect to the evaluation of logical queries Given a logical rule, there are two ways to execute this rule, namely, bottom-up and top-down The *bottom-up* method first evaluates all the predicates in the rule body, and then projects the conjunction of these predicates into the rule head In PNLP, this is simulated by FIR the transition corresponding to the rule The *top-down* methods use the constants (bind-

Symbol	Description
$\overline{F}_t$	Fire transition t, don't consume input tokens
F_t	Fire transition t, and consume input tokens
$F_{t\|(n)}$	Fire transition t, use only tokens marked (n)
F_t^m	Fire transition t, and mark the output token
$\overline{R}_t$	Perturbe t, don't consume input query token
R_t	Perturbe t, consume input query token
$R_{t\|(n)}$	Perturbe t, use the token marked (n)
R_t^m	Perturbe t, mark the output tokens
R_t^c	Perturbe t, produce 'count' of free tokens
$p\|i\| \rightarrow q\|j\|$	Sideway information passing from i^{th} arg of p to j^{th} arg of q
$p\|i\| \overset{(n)}{\rightarrow} q\|j\|$	Sideway inf passing Use a token marked(n)
$p\|i\| \Rightarrow q\|j\|$	Cycle-stealing information passing
$p\|i\| \overset{(n)}{\Rightarrow} q\|j\|$	Cycle-stealing inf Use a token marked(n)
$\sigma(p)$	Select tuples from the relation corresponding to place p

Table 3 4-1

ings) in the query by unifying them with the rule head, and propagating these constants to the rule body, and then evaluating each predicate in the body This corresponds to PRT the transition in PNLP So, given an algorithm for evaluating logical queries in deductive databases, its behavior as modeled by a string in the PNLP language will indicate the class of the algorithm. For example, a pure bottom-up algorithm will result in a string of only FIR operations, while a pure top-down procedure will result in a sequence of PRT followed by FIR, such that no transition is fired unless it is first perturbed

4. Related Works

Most of the algorithms used to evaluate queries in deductive databases use a graph model, such as *the predicate connection graphs* [HN], and *the rule/goal graphs* [U].

The predicate connection graph (PCG) connects the literals that can be unified i.e predicate occurences in the rule body are connected to their occurences in the heads of the rules The PCG is static and displays only the structure aspect of the logic program It does not display by itself the dynamic aspect and the binding propagations through the flow of control Usually, these graphs are used with the top-down strategies

The rule/goal graphs (R/G), which are essentially AND/OR graphs, are also static Each predicate is represented by a relation-node, and each rule is represented by a rule-node Each relation-node is an OR node for all the rule-nodes which correspond to rules with this relation name in the head The rule-node is an AND node for all the relation-nodes corresponding to the predicates in the body Ullman [U] also defines the adorned R/G graphs in which every IDB predicate is represented by a number of relation-nodes corresponding to the different possible binding patterns for this predicate Although adorned R/G graphs can display the propagation of constants, the graph becomes complex, and its size will be an exponential function of the maximum number of arguments in an IDB predicate Together with the idea of capture rules [U], the system is a very general formulation to the problem of query evaluation But unfortunately, R/G graphs support only those 'obvious' capture rules [U]

These graph models are usually tuned heavily to serve specific algorithms and can hardly be used outside the specific algorithm(s) for which they are designed On the other hand, The PNLP model proposed here is formal, and can model almost all the strategies and the flow of control of different algorithms In fact, the motivation for the PNLP model is to use it as a basis for a general inference engine which is able to produce an optimal flow of control for answering queries in deductive databases

5. Modelling the Behavior of Some Query Evaluation Algorithms

In this section, we will use the PNLP to model some of the well known algorithms for recursive query evaluation In order to demonstrate the flow of control of these algorithms, we will use the following example (similar to the same-generation problem [BMSU])

Suppose we have a directed graph with three kinds of edges, u, d, and q, which are stored in the base relations u(x, y), d(x, y), and

q(x, y) respectively where, x and y are nodes in the graph Suppose also that we are interested in finding the nodes n and m that are connected by a special path so that this path can be partitioned into exactly three parts The first part consists of zero or more edges of type u, the second part is exactly one edge of type q, and the third part consists of zero or more edges of type d Moreover the number of edges in the first and the third part are the same Let h(n, m) be such a relation between nodes n and m h can be defined by the following rules

r_1 h(X, Y) - u(X, X1), d(Y, Y1), h(X1, Y1)
r_2 h(X, Y) - q(X, Y)

Now given the query h(a, ?), we want to find all the nodes connected by the special path to node a The graphical representation of the PNLP structure model of the above program is given in Figure 5 1

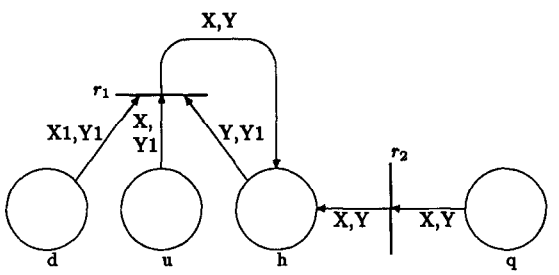

Figure 5 1

The initial markings assumed by the algorithms, discussed in the following sections, differ from one algorithm to another Usually, top-down algorithms consider the query token (a, ?) in place h and tokens in the places corresponding to base predicates as their initial markings Bottom-up algorithms will start with a token in every place corresponding to a base predicate which represent the whole base relation

5.1. Naïve evaluation method [BR]

Naïve evaluation method is a bottom-up method The iterations for evaluating h(a, ?) in our example are*

* h_i denotes the value of h after iteration number i

$$
\begin{aligned}
i=0 \quad & h_0 = q \\
i=1 \quad & h_1 = h_0 \bowtie u \bowtie d \\
i=2 \quad & h_2 = (h_0 \cup h_1) \bowtie u \bowtie d \\
& \\
i=r \quad & h_r = (\bigcup_{j=0}^{j=r-1} h_j) \bowtie u \bowtie d
\end{aligned}
$$

The algorithm stops when $h_i - h_{i-1} = \emptyset$, and the answer is selected from h_i

The control flow of this algorithm can be demonstrated using the PNLP language (refer to Table 3 4-1 for the definitions of symbols) as follows

Initial marking Tokens in q, u, and d representing the EDB relations

Execution sequence

$$
\begin{aligned}
i=0 \quad & \overline{F}_{r_2} \\
i=1 \quad & \overline{F}_{r_1} \\
i=2 \quad & \overline{F}_{r_1}
\end{aligned}
$$

So, the execution string will be $\overline{F}_{r_2}\ \overline{F}_{r_1}\ \overline{F}_{r_1}\ \overline{F}_{r_1}$ Notice that the string consists of only FIR actions which means that the naïve method is a pure bottom-up strategy as mentioned earlier

5.2. Semi-naïve method [BR]

Semi-naïve method tries to cut down the duplicate tuples generated in naïve method The iterations performed by semi-naïve method can be written as

$$
\begin{aligned}
i=0 \quad & h_0 = q \\
i=1 \quad & h_1 = h_0 \bowtie u \bowtie d \\
i=2 \quad & h_2 = h_1 \bowtie u \bowtie d \\
& \\
i=r \quad & h_r = h_{r-1} \bowtie u \bowtie d
\end{aligned}
$$

where at each iteration we only use the new tuples The algorithm will halt when $h_i = \emptyset$, and the result is selected from the union of all h_i's

In PNLP, the sequence of actions describing this method is essentially the same as the sequence of naïve method The difference is that

only the new generated tokens are used for firing In other words, input tokens are consumed (Note that base predicates always have a token) So, the execution string can be written as

$$F_{r_2} F_{r_1} F_{r_1} F_{r_1}$$

5.3. Magic sets [BMSU],[BR]

Magic sets is a term-rewriting optimization method Given a set of rules, the method will generate an equivalent set of adorned rules The new set is divided as "magic rules" and "modified rules" and the evaluation process will be pure bottom-up in two phases

Phase 1 Compute the magic sets from magic rules This phase simulates the passing of bindings in top-down strategies

Phase 2 Use the magic sets computed in phase 1, so that database retrieval can be restricted to only the tuples that have bindings from these magic sets

Since the magic sets method try to simulate top-down flow of control in a bottom-up manner, we will demonstrate its behavior using the new adorned set of rules In the discussion following this section, we will present a PNLP string that operates on the original set of rules and have the same optimization effect as the magic sets method The set of adorned rules corresponding to our example are

Magic rules

t_1 $magic_h^{bf}$(X1) - u(X, X1), $magic_h^{bf}$(X)
t_2 $magic_h^{fb}$(Y1) - d(Y, Y1), $magic_h^{fb}$(Y)
t_3 $magic_h^{bf}$(a)

Modified rules

t_4 h^{bf}(X, Y) -u(X, X1), d(Y, Y1), $magic_h^{bf}$(X) , h^{bf}(X1, Y1)
t_5 h^{bf}(X, Y) - q(X, Y), $magic_h^{bf}$(X)
t_6 h^{fb}(X, Y) - u(X, X1), d(Y,Y1), $magic_h^{fb}$(Y) , h^{fb}(X1, Y1)
t_7 h^{fb}(X, Y) .- q(X, Y), $magic_h^{fb}$(Y)
t_8 $query^{f}$(X) - h^{bf}(a, X)

The relevant part of the PNLP structure of the new set of rules will be as shown in Figure 5 2

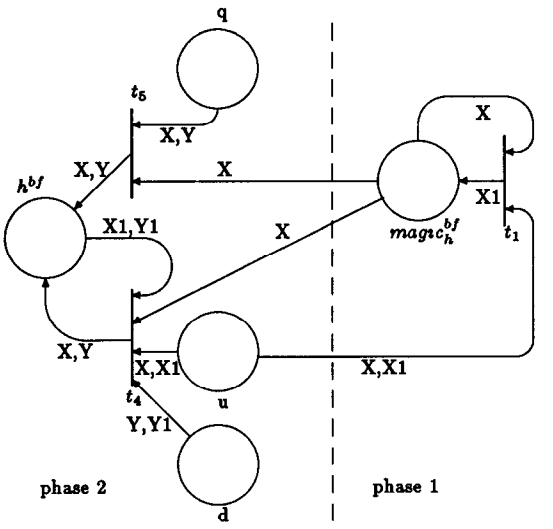

Figure 5 2 PNLP structure of the magic set example

Using semi-naïve or naïve method, the execution will be in two phases In the first phase, only transition t_1 will be active, and initially a token (a) is placed in the place $magic_h^{bf}$ In the second phase only t_4 and t_5 will be active with an initial marking in q, $magic_h^{bf}$, u, and d In terms of PNLP language (Naïve method) this will be

Phase 1 $\bar{F}_{t_1} \bar{F}_{t_1} \bar{F}_{t_1}$ $\bar{F}_{t_1}$
Phase 2 $F_{t_5} \bar{F}_{t_4} \bar{F}_{t_4} \bar{F}_{t_4}$ $\bar{F}_{t_4}$

Notice that the PNLP strings indicate that it is a pure bottom-up strategy

5.4. Henschen-Naqvi algorithm [HN]

HN-algorithm is a top-down algorithm The main idea in the algorithm is to identify a special loop in the predicate connection graph, called *Potential Recursive Loop* (PRL) This PRL is then divided into two parts *determined part*, and *induced part* The role of the PRL is that the only way to obtain an answer to a query is to evaluate around the PRL zero or more times (full cycles), and then through the exit expression So, the algorithm in the i^{th} iteration, will evaluate the determined part i times then, evaluate the exit expression once, and then evaluate the

induced part i times The iterations stop when the determined part evaluation yield $\emptyset$ Notice that the induced part and the determined part depend on the form of the query In our example, we have

Determined part = u(X, X1),
Induced part = d(Y, Y1),
Exit expression = q(X, Y)

Thus, the sequence of DBMS requests generated by HN-algorithm will be equivalent to the following sequence

i=0 q(a, Y)
i=1 $u(a, X1^1) \bowtie q(X1^1, Y1^1) \bowtie d(Y, Y1^1)$
i=2 $u(a, X1^1) \bowtie u(X1^1, X1^2) \bowtie q(X1^2, Y1^2) \bowtie d(Y1^1, Y1^2) \bowtie d(Y, Y1^1)$

In fact, the algorithm saves the result of evaluating the determined part in the i^{th} iteration and uses it in the next iteration The flow of control in PNLP language will be as follows

initial marking h(a, ?), u, d, and q (at the beginning of each iteration)
Execution sequence

i=0 $R_{r_2}\ \sigma(q)\ F_{r_2}$
i=1 $R_{r_1}\ \sigma(u)\ u[2]\rightarrow h[1]\ R_{r_2}$
$\sigma(q)\ F_{r_2}\ h[2]\rightarrow d[2]\ \sigma(d)\ F_{r_1}$
i=2 $R_{r_1}\ \sigma(u)\ u[2]\rightarrow h[1]\ R_{r_1}\ \sigma(u)\ u[2]\rightarrow h[1]\ R_{r_2}$
$\sigma(q)\ F_{r_2}\ h[2]\rightarrow d[2]\ \sigma(d)\ d[1]\Rightarrow d[2]\ \sigma(d)\ F_{r_1}$

Notice that the iteration will halt when $\sigma(u)$ in the determined part produces $\emptyset$

We can also simulate the optimization done in the process for evaluating the determined part The new sequence will be
Initial marking h(a, ?), u, d, q
Execution Sequence

i=0 $\overline{R}_{r_2}\ \sigma(q)\ F_{r_2}$
i=1 $R_{r_1}^c\ \sigma(u)\ u[2]\rightarrow h[1]\ \overline{R}_{r_2}\ \sigma(q)\ F_{r_2}\ h[2]\rightarrow d[2]$
$\sigma(d)\ F_{r_1}$
i=2 $R_{r_1}^c\ \sigma(u)\ u[2]\rightarrow h[1]\ \overline{R}_{r_2}\ \sigma(q)\ F_{r_2}\ h[2]\rightarrow d[2]$
$\sigma(d)\ d[1]\Rightarrow d[2]\ \sigma(d)\ F_{r_1}$

Here we use $\overline{R}_{r_2}$ in order to save the last result, and use $R_{r_1}^c$ to produce a number of free tokens equal to the count c of transition r_1 This enables the net to evaluate $\sigma(d)$ (induced part) the correct number of times

5.5.Counting method [BMSU], [BR], [SZ]

Although the counting method first appeared in the literature as a modification of the magic sets technique [BMSU], the control flow of the method is essentially the same as that of the HN-algorithm This can be observed from the corresponding PNLP model The difference between the two algorithms is due to the decision about the trade-off between storage and computation From the previous section, each iteration of the HN-algorithm consists of two phases The first is the evaluation of the determined part (the computation is optimized in this phase), and the second is the evaluation of the induced part (the computation is not optimized) A set of answers will be produced after each iteration On the other hand, the counting method will evaluate the determined part for all the iterations in the first phase (saving the results of each iteration, and these are called the *counting sets*), and then evaluate the induced part in the second phase The result will be produced only after the last iteration in the second phase This scheme allows the counting method to optimize also the computation in the second phase With this view in mind, the PNLP control sequence of the counting method can be as follows

Initial marking h(a,?), u, d, and q
Phase 1 (Generating counting sets)

$i=0$ $\quad \overline{R}_{r_2}\, \sigma(q)\, F_{r_2}^m$

$i=1$ $\quad R_{r_1}\, \sigma(u)\, u[2] \rightarrow h[1]\, \overline{R}_{r_2}\, \sigma(q)\, F_{r_2}^m$

$i=2$ $\quad R_{r_1}\, \sigma(u)\, u[2] \rightarrow h[1]\, \overline{R}_{r_2}\, \sigma(q)\, F_{r_2}^m$

$i=n-1$ $\quad R_{r_1}\, \sigma(u)\, u[2] \rightarrow h[1]\, \overline{R}_{r_2}\, \sigma(q)\, F_{r_2}^m$

$i=n$ $\quad R_{r_1}\, \sigma(u)$

Phase 2 Initial marking from phase 1

$i=0$ $\quad h[2]^{(\underline{n-1})}\, d[2]\, \sigma(d)$

$i=1$ $\quad (d[1] \Rightarrow d[2],\ h[2]^{(\underline{n-2})}\, d[2])\, \sigma(d)$

$i=2$ $\quad (d[1] \Rightarrow d[2],\ h[2]^{(\underline{n-3})}\, d[2])\, \sigma(d)$

$i=n-1$ $\quad (d[1] \Rightarrow d[2],\ h[2]^{(\underline{1})}\, d[2])\, \sigma(d)$

$i=n$ $\quad F_{r_1|(1)}$

Notice that the operations in the second phase which are enclosed in brackets use only one free token That is, $(d[1] \Rightarrow d[2],\ h[2]^{(\underline{n-2})}\, d[2])$ is equivalent to $(d[1] \cup h[2]) \rightarrow d[2]$

6. Discussion

In this section, we discuss the query evaluation algorithms that are considered in the previous section in terms of their PNLP model, and demonstrate the functionality of the PNLP model That is, we discuss the similarities and differences of these algorithms that are made more apparent by their PNLP models.

The flow of control in the naïve method described in section 5 1, implies that tokens are not consumed upon firing of transition r_1 (i e $\overline{F}_{r_1}$), so tokens will accumulate in place h, and the cost for firing (which is a JOIN operation) will increase as we iterate Also, tokens that are generated in the i^{th} iteration will also regenerated in all the subsequent iterations This increases the cost for redundancy checking The sequence of control chosen by the naïve method will never lead to a halting PNLP, and the only way for the algorithm to stop is to provide external check for the termination condition after each firing of r_1

The semi-naïve method avoids the above problems by choosing to consume the input token upon firing (i e F_{r_1}) This also leads to a halting PNLP, when the database relations are finite That is because when the result of F_{r_1} is empty, the place h will no longer have tokens and consequently, r_1 will not be enabled This makes the semi-naïve method much more efficient and the volume of data (tokens) circulating in the net is reduced

In the magic sets method the sequence $\overline{F}_{t_1}$ $\overline{F}_{t_1}$ appearing in the first phase, can be considered as a sequence of selections from the relation u Instead, we can perform the first phase in a top-down fashion using the original set of rules Notice that the new sequence will be as efficient as the old one, since it corresponds to the same series of selections The advantage is that, we need not deal with the set of adorned rules which is much larger than the original set of rules Thus, the following sequence can be considered as another approach which has the same effect as the magic sets method but without any re-writing of the rules

phase 1

Initial marking h(a, ?), u, d, and q

Execution sequence $R_{r_1}\sigma(u)\, u[2] \rightarrow h[1]\, R_{r_1}\, \sigma(u) \rightarrow h[1] \quad R_{r_1}\, \sigma(u)$ *

phase 2

Initial marking d and q (base relations), and tokens in u from phase 1

Execution sequence $F_{r_2}\, \overline{F}_{r_1}\, \overline{F}_{r_1} \quad \overline{F}_{r_1}$

The PNLP models of Counting method clearly demonstrates the relationship between Counting and HN-algorithm That is the only difference between the HN-algorithm and the Counting method is that, the Counting method also does optimization in the induced part This also explains the relative performance of the HN-algorithm and the Counting method presented in [BR] That is, in [BR] several test programs are used, and using a cost model, for each method and for each query the cost of computing the query is analitically computed for the different sets of data From the results of this cost computation, it can be observed that in each

* Stop when last selection yields $\emptyset$

case where the HN-algorithm and the Counting method perform the same, the test program used does not have an induced part, and for the cases where the Counting method performs better than the HN-algorithm, the test program used has an induced part

Before concluding the discussion, we should mention that the PNLP strings given here to describe the behavior of these algorithms are not the only possible sequences This is due to the inherent non-determinism of the PNLP model

7. Conclusion and Future Work

We present a non-deterministic modelling technique for evaluating logical queries in deductive databases The model is a colored extension to Petri Net model We call this model the Petri Net model for Logic Programs (PNLP) The structure aspect of this model captures the structure of Horn-clause logic programs, and the execution aspect allows the power of non-deterministic modelling to the flow of control for evaluating the queries, especially the recursive queries The model also illustrates the binding propagation process during the query evaluation The PNLP is motivated by a research effort for developing an automata for evaluating logic queries with the ability to choose an optimum flow of control This automata should use both the static characteristics and the existing knowledges about the database and the dynamic facts discovered during execution

We demonstrate the power and the generality of the PNLP model by using it for the modelling of several well-known algorithms Also in these examples, we use the model to identify the basic differences on both the data flow and the control flow, and generally their impact on the performance The description of the flow of control and the data flow in terms of the PNLP language is concise We are also able to use the PNLP to produce, as an example, a new version of the magic sets method which does not require the rewriting of the original set of rules into a new set of adorned rules (which is usually much larger set)

Although, we present here only the model and its modelling capacity, currently we are investigating the following topics related to the PNLP model

- Methods for recognizing parallelism in the model,
- Relationships between the structure of a given PNLP and the execution schedule, in order to obtain an efficient flow of control for special classes of Horn clauses
- Formal definition of a full automata that can efficiently execute the model, and its implementation issues

References

[A] H Aly, "Petri Nets application to queueing systems" M Sc thesis, Alexandria university, Egypt, 1983

[BMSU] F Bancilhon, D Maier, Y Sagiv, and J Ullman, "Magic sets and other strange ways to implement logic programs" ACM PODS, 1986

[BR] F Bancilhon, R Ramakrishnan, "An amateurs's introduction to recursive query processing strategy" ACM SIGMOD, 1986

[CJ] C Choppy, C Johnen, "PETRIREVE Proving Petri Net properties with re_writing systems" in Lecture Notes in Computer Science No 202, Springer-Verlag, 1985

[GM] H Gallaire, J Minker, Logic and databases, Plenum Press, New York, 1978

[HN] L Henschen, S Naqvi, "On compiling queries in recursive first order databases" JACM, Vol 31,No 1, 1984

[K] R Kowalski, Logic for problem solving, North-Holland 1979

[P] J Peterson, Petri Net theory and the modelling of systems, Prentice-Hall, 1981

[SZ] D Sacca, C Zaniolo, "On the implementation of a simple class of logic queries for databases" ACM PODS, 1986

[U] J Ullman, "Implementation of logical query languages for databases", ACM TODS, Vol 10, No 3, 1985