World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:11, 2007

# New Approach for Manipulation of Stratified Programs

Amel Grissa-Touzi, Chadlia Jerad, and Habib Ounelli

***Abstract*** - Negation is useful in the majority of the real world applications. However, its introduction leads to semantic and canonical problems. We propose in this paper an approach based on stratification to deal with negation problems. This approach is based on an extension of predicates nets. It is characterized with two main contributions. The first concerns the management of the whole class of stratified programs. The second contribution is related to usual operations optimizations on stratified programs (maximal stratification, incremental updates …).

***Keywords*** - stratified programs, stratification, standard model, update operations, SEPN formalism.

## I. Introduction

Logical programming constitutes a powerful tool for the treatment of several problems in particular in artificial intelligence [1] and deductive databases [2].

Many real world applications need the use of negation for modeling negative information. Negation introduction leads to several problems, in particular the definition of a canonical semantics for these programs [3], [4]. Several works showed that under certain syntactic restrictions, it is possible to define a canonical semantics of normal programs. This leads to stratified programs [3], [4]. The approach based on stratification received attention on behalf of the researchers. Unfortunately, implementation aspects, in particular, representation structures and manipulation algorithms, were completely neglected.

We propose in this paper an original extension of predicates nets (EPN), noted SEPN, as representation structure of stratified programs. In addition to their formal aspect, EPN nets proved their efficiency in modeling knowledge bases, in particular in artificial intelligence [1], deductive databases [2] and expert systems [5]. However, EPN algorithms require non acceptable execution time in case of large programs.

We establish a correspondence between the SEPN and stratified programs. This correspondence was used for building an efficient implementation of this type of programs and countering EPN problems.

Manuscript received March 31, 2005.

A. G.T. is with the Department of Telecommunication and Information Technologies of National School of Engineers of Tunis, Tunisia (e-mail: amel.touzi@enit.rnu.tn).

C. J. is with the Department of Electrical Engineering of National School of Engineers of Tunis, Tunisia (e-mail: jerad.chadlia @ gawab.com).

H. O. is with the Department of Computer Science of Faculty of Sciences of Tunis, Tunisia (e-mail: habib.ounelli@fst.rnu.tn)

Our approach has two main parts: (1) SEPN and (2) manipulation algorithms of stratified programs. Due to space limitation, we devote this paper to the first part. The second part is the subject of paper [6].

The remainder of the paper is organized in six sections. Section 2 presents basic concepts of stratified programs. Section 3 describes our approach based on SEPN. The correspondence between stratified programs and SEPN is presented in section 4. Section 5 presents the advantages of SEPN use and an example. Section 6 concludes the paper and gives some extensions of our work.

## II. Basic concepts

We suppose known basic concepts of logical programming [7]. We recall briefly, in this section, basic notions related to stratified programs.

### A. *Stratified Programs*

Negation introduction leads to many problems, in particular the definition of a canonical semantics for these programs [3], [4]. Several works showed that stratification is a possible solution to the treatment of negation problems [3], [4]. Indeed, under certain syntactic conditions, the problem of the choice of a model can be solved by dividing the program into elements called stratum. This decomposition is made in order to allow the use of negative literals only if all their logical consequences are already deduced in the model. Thus, we are able to apply closed world assumption. We present in what follows basic definitions related to stratified programs [4], [8].

Let $P$ be a logical program. A predicate symbol $q$ definition is the set of all the clauses of the program $P$ having $q$ at the head of the clause. A program $P$ is stratifiable if there is a partition $P = P_1 \cup \ldots \cup P_n$ (where $P_1$ can be empty), called stratification of $P$ such as for each $i = 1, 2, \ldots, n$, we have the following properties:

- if a predicate symbol is positive in $P_i$ then its definition is contained in $\cup_{j \le i}\ P_j$.
- if a predicate symbol is negative in $P_i$ then its definition is in $\cup_{j < i}\ P_j$.
- Each $P_i$ is called a strata.

The dependency graph of a program $P$ ($Dp$) is composed of a set of nodes connected by arcs. Each node represents a predicate of $P$. The arc matching $r$ and $q$, noted $(r, q)$, belongs to $Dp$ if there is a clause in $P$ using $r$ in its head and $q$ in its body. We say $r$ refers to $q$. If a predicate $q$ appears positively (respectively negatively) in the body then $(r, q)$ is called positive (respectively negative).

A program is stratifiable if and only if, its dependency graph does not contain any circuit containing a negative arc [4], [8].

A stratifiable program can have several stratifications [8]. This property is used in the remaining of the article without demonstrating it here.

A stratification $P = P_1 \cup \ldots \cup P_n$ is maximal stratification if each strata cannot be decomposed into different stratum. Let $P_i$ be a stratum and $M$ a set of facts, we denote by $SAT(P_i, M)$ the saturation of $M$ by $P_i$, which is the set of facts obtained by the closing of $M$ under the clauses of $P_i$.

Let $P = P_1 \cup \ldots \cup P_n$ be a maximal stratification of $P$, we define the standard model of the program $P$ ($M_P$), by proceeding in recursive way the following operation [8]:

$$M_1 = SAT(P_1, \varnothing)$$
$$\ldots$$
$$M_P = M_n = SAT(P_n, M_{n-1})$$

The *Mp* model has three properties, which are (1) *Mp* does not depend on the stratification of *P*, (2) *Mp* is a standard model of *P* and (3) *Mp* is a model of completion of *P*.

In general, *SAT* function depends on the order of program clauses. However, this is not the case for a stratum of *P*. Indeed, we apply the closed world assumption directly because the definition of negative literals is on a strictly lower level. Thus, there is not possibility of deducing new facts relating to this literal.

## B. Update of a stratified program

This subsection is dedicated to the study of update operations on stratified programs.

An update operation is the removal or the addition of a fact or a rule of a program [8]. An update operation is accepted if the following conditions are verified: (1) any constant, which does not belong to the language describing the program, can not be introduced, (2) the inserted clause must be "Range-Restricted", this means that the variables appearing in the head of the clause appear in its body and (3) the obtained program remains stratified.

An update operation transforms a program *P* into a program *P'*. Consequently, the $M_P$ model associated to *P* is transformed into a model *Mp'* associated to *P'*. The new updated model *Mp'* can be completely different from *Mp*. In fact, *Mp'* may contain facts not belonging to *Mp*, without being an over set. In general, the new *Mp'* model computation consists on the removal and addition of facts.

The automatic determination of update operations' results is delicate and leads to performance problems related to execution time. These problems are directly related to the modeling approach of the program.

We introduce, now, the effect of update operations on stratifiability and *Mp* model computation.

### 1) Update operations effects on stratifiability

After an update operation, we should, first of all, check that stratifiability property of the program. Then, it is necessary to define the relation between the maximal stratification of the initial program *P* and the maximal stratification of the resulting program *P'*. All the possibilities cases are presented in tables I and II.

TABLE I
EFFECTS OF ADDITION UPDATE OPERATIONS ON STRATIFIABILILY

| Update operation | Consequences | Strati-fiability |
|---|---|---|
| Fact q(a,b): the predicate q is not defined in the program P. | Stratum creation. | Yes |
| Fact q(a,b): the predicate q is already defined in the program P. | Stratum modification. | Yes |
| Clause: the predicate of the head appears for the first time. | Stratum creation. | Yes |
| Clause: the predicate of the head is already defined. | If the program is still stratifiable :<br>- Stratum modification.<br>- Or fusion of several strata | Depends on cases |

TABLE II
EFFECTS OF REMOVAL UPDATE OPERATIONS ON STRATIFIABILILY

| Update operation | Consequences | Strati-fiability |
|---|---|---|
| Fact : the fact is composing a stratum. | Stratum removal. | Yes |
| Fact : the fact belongs to a stratum composed of several clauses. | Stratum modification. | Yes |
| Clause: the clause composes the stratum. | Stratum removal. | Yes |
| Clause: the clause belongs to a stratum composed of several clauses. | - Stratum modification.<br>- Or the stratum is splited into several strata. | Yes |

Several works used predicates sets, called supports [4], in order to compute the resulting model *Mp'* after an update operation. These supports are used to find out the facts to be removed from the model *Mp* after the update operation. The supports choice should minimize the number of facts migrations and the maintenance cost of these supports. The ideal situation is to have a support which determines exactly the facts which must be removed from model to avoid facts migrations [3], [8].

### 2) Update operations effects on the standard model

After studying update operations on stratified programs, we distinguished two cases: (1) the addition or removal of an explicit deduced fact has no effect on program stratifiability or standard model and (2) the removal operation of such a fact is not even significant. The explicit addition or removal of not deduced fact or a clause may affect the program stratifiability and its standard model. In this last case, an update operation affects a number of deduced facts (known as induced updates) because of clauses. The main concept concerning the management of induced updates is their determination in an efficient way.

## III. SEPN FORMALISM

## A. SEPN nets

The Stratified Extended Predicates Nets, noted SEPN [9], [10], are based on an extension of RPENS nets [3], [9], [10], [11], enriched with two new concepts: attribution colors to transitions and the firing process of a transition.

In this section, we describe the SEPN formalism. For further details concerning this approach, readers can refer to [9], [10], [11].

An SEPN is defined by:

- A quintuple *N = (P, T, C, V, K)*, where *P*, *T*, *C*, *V* and *K* are respectively the set of places, the set of transitions, the set of colours, the set of variables and the set of constants.
- Two relations $\alpha$ and $\beta$, where $\beta$ is a finite subset of $T \times P$ which elements are called unsigned arcs and $\alpha$ is a finite subset of $\{+,-\} \times P \times T$ which elements are called signed arcs ($\alpha$+ positive arcs set and $\alpha$- negative arcs set).
- Two applications $I\alpha$ et $I\beta$ defined by:
    - $I\alpha : \alpha \rightarrow Z[V \cup K]$
    - $I\beta : \beta \rightarrow Z[V \cup K]$

  where $Z[V \cup K]$ is the set of finite formal combinations of $V \cup K$ elements.
- A set *Garde*, where *Garde(t)*, *t* being a transition, imposes firing conditions between tokens contained in input places.
- A bijective application *Cl* from *T* to *C*, which associates a color to each transition.

## B. Dynamic aspect of the SEPN

In a SEPN net, tokens are colored [9], [10], [11]. A colored token is an element of the set $K^n \times P(C)$, where *P (C)* is a set of parts of *C*. A colored token *j* has then this form: $j = ((x_1, x_2, \ldots, x_n), Col)$ where $(x_1, \ldots, x_n)$ is the argument of *j* *(arg(j))* and *Col* its path *(path(j))*. The path is a set of colors saving the history deduction.

We define the following operations on the elements of the set $K^n \times P(C)$:

- Equality: $j = j' \Leftrightarrow arg(j) = arg(j')$ and $path(j) = path(j')$
- Order relation $\leq$: $j \leq j' \Leftrightarrow arg(j) = arg(j')$ and $path(j) \subseteq path(j')$
- Subtraction: if $arg(j) = arg(j')$ then $j - j' = (arg(j), path(j) \setminus path(j'))$.
- A token of the form $j = (arg(j), \{\emptyset\})$ is called neutral token.

Consequently, we have these two results (1) the subtraction to a token from itself gives a neutral token and (2) the addition operation is idempotent ($j + j = j$).

Let *R* be an SEPN net and *Mo* a function from *P* to $K^n \times P(C)$. The function *Mo* is called initial marking of *R*. This function associates, initially, to each place of *R* a finite set of colored tokens. The SEPN marking *M* is defined by the association to each place a finite set of tokens.

The transition firing process in SEPN is different from ordinary Petri nets. In fact, the production of new tokens happens without removing the tokens used while firing the transition. We make a distinction between valid transitions from fireable transitions.

If the new token already exists in the destination place, it will not be regenerated. This transition is then fireable but not valid. If the token does not exist in the output place, the transition is valid and is fired. By this way, an SEPN can not contain double tokens. Each place *p* is $k^{m(p)}$ bounded, where *m(p)* is the marking of *p* and k is the cardinality of the set *K*.

TABLE III
CORRESPONDENCE BETWEEN STRATIFIED PROGRAM S AND RPES

| Stratified program | RPES |
|---|---|
| Program alphabet | $V \cup K$ |
| A predicate p | A place p |
| A fact q(a,b) | A neutral token in the place q ((a,b),{∅}) |
| A clause r | A transition t |
| Link between the predicates' variables and the clause body | Garde of the transition |
| Clause body | Sub-sets of α+ and α- with their respective labels |
| Head clause | An element of β with its label |
| Standard model of the program | Net marking |
| Program stratification | Net stratification |
| Program strata | Net strata |
| Addition or removal of a fact | Addition or removal of a token |
| Addition or removal of a clause | Addition or removal of a transition and its related arcs |

## IV. CORRESPONDENCE BETWEEN STRATIFIED PROGRAM AND SEPN NETS

The SEPN formalism allows us to build efficient algorithms for checking stratifiability, determination of the maximal stratification, the computation of the standard model, and management of update operations on facts and clauses (explicit and induced updates). Indeed, we established a correspondence between the SEPN formalism and stratified program. This correspondence is presented in table III. Due to space limitations, we do not demonstrate this correspondence.

## V. SEPN ADVANTAGES

### A. Stratifiability study and maximal stratification determination using SEPN

Stratifiability checking is released after the definition of the logical program and after each update operation. This consists on detecting the presence of a negative circuit in the SEPN. Once the stratifiability of a program is checked, it is necessary to find out the maximal stratification in order to compute the standard model. For this purpose, we give the following definitions:

**Definition 1**: Let *R* be a SEPN and $V = \{p1, T1\ldots, pn, tn\}$ (such as $\{p1\ldots, pn\} \in P$ and $\{T1.., tn\} \in T$) a strongly connected component of *R*. We define a stratum of *R* as a strongly connected component *V* of *R* with one of following conditions:

- $Card(V) > 1$
- $Card(V) = 1$ and $V = \{p\}, p \in P$, with $M(p) \geq 0$ or $\exists\, Ti \in T \,/\, (Ti, p) \in \beta$. Where *M(p)* is the marking of the place *p*.

After determining the SEPN strata, we put an order on them. We introduced, for this purpose, the concept of stratified reduced graph.

**Definition 2**: Let *G* be the set of the SEPN strata. The stratified reduced graph of the SEPN is the graph *Gr = (X, U)*, where:

- Each node of *X* represents a stratum.
- And $U = \{(S_i, S_j),\ i \neq j \mid \exists\, p \in S_i \text{ and } T \in S_j \mid (p,t) \in P\}$, where *U* is a finite set of arcs connecting strata.

An arc of the stratified reduced graph is positive (respectively negative) if there is a positive arc (respectively negative) in the SEPN relating a place of $S_i$ and a transition from $S_j$. The stratified reduced graph of an SEPN does not contain circuits. It represents dependences between strata.

**Definition 3**: Let *R* be a stratified SEPN and $V_1,\ldots, V_n$ a maximal stratification of *R*. In the stratified firing process, the firing of a transition in $V_i$, $i \in [1..n]$, starting from a given marking of SEPN, can be made only if the firing of all the transitions belonging to $V_j$, $j \leq i-1$, is done.

## B. Example

Let us consider the following program *P*:

$$P = \begin{cases} F(f, c) \leftarrow ; \\ G(n, p) \leftarrow ; \\ A(x, z) \leftarrow C(y, z); \\ C(x, z) \leftarrow A(y, z), F(x, y); \\ D(z, y) \leftarrow G(z, y), not\ (C(x, z)) ; \\ G(x, z) \leftarrow D(x, y), G(y, z); \end{cases}$$

Figure 1 shows the SEPN corresponding to *P*. The program is stratifiable since its SEPN does not contain recursion through negation.

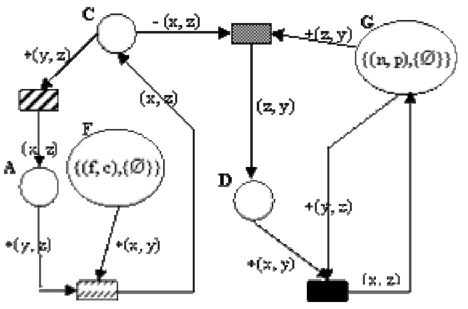

Fig. 1. SEPN net of the program P

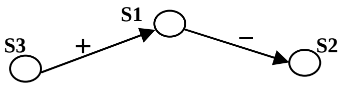

Fig. 2. Reduced graph of P

The reduced graph of *P* is shown in figure 2. It shows the maximal stratification of P, which is composed of these strata:

$$S1 \begin{cases} A(x, z) \leftarrow C(y, z); \\ C(x, z) \leftarrow A(y, z), F(x, y); \end{cases}$$

$$S2 \begin{cases} G(n, p) \leftarrow ; \\ D(z, y) \leftarrow G(z, y), not(C(x, z)) ; \\ G(x, z) \leftarrow D(x, y), G(y, z); \end{cases}$$

$$S3 \{ F(f, c) \leftarrow ;$$

## C. Update operation optimization in SEPN

### 1) Update operation optimization of facts

The addition of a token in a place *p* may lead to: (1) the addition of other tokens in the places related positively to *p* and (2) the removal of tokens from the places negatively related to *p*. In opposition, the removal of a token of a place *p'* may lead to: (1) the addition of tokens in the places negatively related to *p'* and (2) to the removal of tokens from the places positively related to *p'*.

The use of colored tokens has the advantage of saving the deduction history. This allows us to recognize the transitions used in the firing process. Thus, it is easy to reduce facts migration. In fact, the tokens that do not contain the transition's color related to the place, in their paths, will not be touched.

### 2) Update operation optimization of clauses

The update of a clause is equivalent to the update of a transition in the PRES net. Knowing the corresponding sub-net of the clause and the stratified reduced graph, we find out, directly, if the program remains stratifiable and its new maximal stratification. After this step, the problem is reduced to facts update, which is already solved.

## VI. Conclusion

We proposed in this paper an approach, based on the SEPN, for the manipulation of a large class of normal programs, which is stratified programs. This class of logical programs is very useful in several fields, in particular artificial intelligence and deductive databases.

This approach covers all the aspects related to stratified programs: data structures, stratifiability checking, maximal stratification determination and incremental updates, knowing stratifiability and the stratification properties on the initial program. This approach was implemented and validated with the STRPRO tool [6]. Obtained results are encouraging for programs composed of, nearly, thirty rules.

In the future, we plan to study the algorithms complexity in case of programs with significant clauses number (hundreds or more).

## References

[1] J. L. Laurière, "Intelligence artificielle, résolution de problème par l'homme et la machine", (Ed) Eyrolles, 1986.
[2] G. Gardarin, "Bases de Données Objet et Relationnel", (Ed) Eyrolles, 2000.
[3] A. Grissa-Touzi, "Contribution à l'Etude, à la Conception et au Prototypage des Bases de Données Déductives", Ph. D. thesis. Dept. of Computer Science, Faculty of Sciences of Tunis, Tunisia, 1994.
[4] G. Jager and R. Stark, "The defining power of stratified and hierarchical logic programs", *Journal .of Logic Programming*, 1993, pp. 55-77.
[5] H. Farreny, "Les systèmes experts principes et exemple", (Ed) Cepadues, Novembre, 1986.
[6] C. Jerad, A. Grissa-Touzi and H. Ounelli, "STRPRO tool for Manipulating Stratified Programs Based on SEPN", submitted for publication in AISC 2005.
[7] J.W. Lloyd, "Fondement de la Programmation logique", (Ed) Eyrolles, Paris, 1988.
[8] R. K. Apt and H. A. Blair, "Arithmetic Classification of Perfect Models of Stratified Programs", Fundamenta Informaticae, vol. 14, 1991, pp. 339-343.
[9] A. Grissa-Touzi, C. Jerad and K Barkaoui, " Nouvelle Approche pour la Définition et la Manipulation de la Négation par les Programmes Stratifiés", Maghrebian Anales of Engineers, vol 19, N°1, 2005.
[10] C. Jerad, " Outil d'Analyse des Bases de Données Déductives Formulées à l'Aide des Réseaux à Prédicats Etendus Stratifiés", Master memory, Dept. of Electrical Engineering, National School of Engineers of Tunis, Tunisia, July, 2003.
[11] A. Touzi and K. Barkaoui, "Un Formalisme de Modélisation et d'Optimisation des Bases de Données Déductives Basé sur la Théorie des Réseaux de Petri de Haut Niveau", 2nd Maghrebine Conference on Software Engeneering and Artificial Intelligence, Tunis, 1992, p 99-115.