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

STRPRO Tool for Manipulation of Stratified Programs Based on SEPN

Chadlia Jerad, Amel Grissa-Touzi, and Habib Ounelli

Abstract - Negation is useful in the majority of the real world applications. However, its introduction leads to semantic and canonical problems. SEPN nets are well adapted extension of predicate nets for the definition and manipulation of stratified programs. This formalism is characterized by two main contributions. The first concerns the management of the whole class of stratified programs. The second contribution is related to usual operations optimization (maximal stratification, incremental updates …). We propose, in this paper, useful algorithms for manipulating stratified programs using SEPN. These algorithms were implemented and validated with STRPRO tool.

Keywords - stratified programs, update operations, SEPN formalism, algorithms, STRPRO.

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, used 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.

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 second part. The first 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 the SEPN formalism. Manipulation algorithms of stratified programs through SEPN are presented in section 4. Section 5 is devoted to the description of STRPRO tool. Section 6 concludes the paper and gives some extensions of our work.

Manuscript received March 31, 2005.

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

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).

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

II. Basic concepts

We suppose known basic concepts of logical programming [7].

A. Stratified programs

We recall briefly, in this section, basic notions related to stratified programs [3], [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 $\cup_{j \leq i} P_j$.
- 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 stratum.

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].

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]:

$M_1 = SAT(P_1, \varnothing)$

…

$M_P = M_n = SAT(P_n, M_{n-1})$

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

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 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. Our approach, based on the SEPN [6], is an efficient attempt to resolving these problems.

Effects of update operations on stratifiability and standard model are presented in [6].

III. SEPN Formalism

In this section, we describe the SEPN formalism. For further details concerning this approach, readers can refer to [6], [9] and [10].

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 α and β, where β is a finite subset of $T \times P$ which elements are called unsigned arcs and α is a finite subset of $\{+,-\} \times P \times T$ which elements are called signed arcs (α+ positive arcs set and α- 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.

In a SEPN net, tokens are colored [9], [10], [[6]. 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.

Dynamic aspects and transition firing process are described in [6].

IV. Main Algorithms

Using the correspondence between stratified programs and SEPN established in [6], we present in this section main algorithms for manipulation of stratified programs.

For this purpose, we adapted certain known algorithms on graphs manipulation, in particular Tarjan algorithm for the determination of strongly connected components [11].

1) Stratifiability study

Stratifiability study goes back to the detection of negative circuits' existence in the net.

2) Maximal stratification determination

The maximal stratification determination of stratifiable programs amounts to identifying the strongly connected components of the SEPN. These components correspond to program strata.

3) Standard model computation

The standard model computation of the stratified programs goes back to applying *SAT* function. We obtain the stable marking of the SEPN net by performing stratified firing of all transitions [6]. Stratified firing depends on strata order in the reduced graph, but does not depend on transitions order in the same stratum. Figure 1 shows the algorithm of the procedure *StratifiedFiring* that performs the stratified firing of a RPES.

```
Procedure SratifiedFiring()
Begin
  For (each stratum sti) Do
    Ver = false;
    While (Var == false) Do
      For (each transition tij ∈ sti) Do
        Fire (tij);
      EndFor
    EndWhile
  EndFor
End.
```

Fig. 1 : Algorithm of the procedure *StratifiedGeneration*

4) Non deduced fact update

The update of a non deduced fact [6] is equivalent to the update of a neutral token (addition or removal). We recall from [6] the following conclusions. 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'*.

Two procedures are used for this purpose *AddTooken* and *RemoveToken*. *AddToken(p, j)* procedure allows the addition of a neutral token *j* to the place *p* passed as parameter, the update of maximal stratification and the update of the standard model. *RemoveToken(p, j)* procedure allows the removal of a token *j* from the place *p* passed as parameters if the token j is not a deduced one, the update of maximal stratification and the update of the standard model.

These two procedures call *UpdateIncPlace(p)* and *UpdateDecPlace(p)* procedures:

- *UpdateIncPlace(Place p)* : this procedure (Fig. 2) is released when the marking of the place *p*, passed as parameter, has increased. The marking of the places related positively to *p* will increase; this is done by firing related transitions. The marking of the places related negatively to p will decrease. This procedure is recursive and calls the two procedures *UpdateDecPlace* et *ModifyThenDelete*.
- *UpdateDecPlace(Place p)* : this procedure (Fig. 3) is released when the marking of the place *p*, passed as parameter, has decreased. The marking of the places related negatively to p will increase; this is done by the firing the related transitions. The marking of the places related positively to p will decrease. This procedure is also recursive and calls the two procedures *UpdateIncPlace* et *ModifyThenDelete*.
- *ModifyThenDelete(Transition t)* : this procedure (Fig. 4) updates history paths of the tokens contained in the output place of the transition *t*. If the path of a token becomes empty, the token is removed from the list of tokens of the place.

```
Procedure UpdateIncPlace (Place p)
Begin
  For (each ti ∈ Inc(p)) Do
    Fire (ti);
    pi = outputPlaceOf(ti);
    If (pi increased) Then UpdateIncPlace(pi);
  EndFor
  For (each tj ∈ Dec(p)) Do
    ModifyThenDelete(tj);
    Fire(tj);
    pj = outputPlaceOf(tj);
    If (pj decreased) Then UpdateDecPlace(pj);
  EndFor
End.
```

Fig. 2: Algorithm of the procedure *UpdateIncPlace*

```
Procedure UpdateDecPlace (Place p)
Begin
  For (each ti ∈ Dec(p)) Do
    Fire (ti);
    pi = outputPlaceOf(ti);
    If (pi increased) Then UpdateIncPlace(pi);
  EndFor
  For (each tj ∈ Inc(p)) Do
    ModifyThenDelete (tj);
    Fire (tj);
    pj = outputPlaceOf(tj);
    If (pj deceased) Then UpdateDecPlace(pi);
  EndFor
End.
```

Fig. 3 : Algorithm of the procedure *UpdateDecPlace*

```
Procedure ModifyThenDelete (Transition t)
Begin
  p = outputPlaceOf(t);
  For (each token j ∈ p) Do
    If (color(t) ∈ path(j)) Then
      path(j) = path(j) \ color(t);
    If (path(j)={∅}) Then Destroy(j);
  EndFor
End.
```

Fig. 4: Algorithm of the procedure *ModifyThenDelete*

5) Clauses update

Clause update is more delicate than fact update, since it can affect the program stratifiability. Clause removal goes back to the removal of the corresponding transition and its related arcs, the stratification update and the standard model update. The addition of a rule amounts to testing, first of all, if the transition does not create recursion through negation in the dependency graph. In case of validity, the transition is added, then the reduced graph is updates, and, finally, the standard model is updated.

V. STRPRO TOOL

In order to help understanding the stratification concept, we have built the tool STRPRO [10]. This tool is plate-form independent since it is developed with Java.

First, we selected an open source tool *PetriTool* [12], used in editing and simulating ordinary Petri nets. Then, we made up required modifications to adapt it to SEPN concept. After that, we added necessary modules for the manipulation of stratified programs. Finally, we added layers of graphical user interfaces.

STRPRO interface (Fig. 4) is composed of a *Menu Bar* and four panels. The program edition is done in the *Edition panel*. The *Gr Panel* holds the reduced graph. The *Strat Panel* contains the composition of each stratum. The *Status Panel* displays messages to users (stratification result, compilation results…).

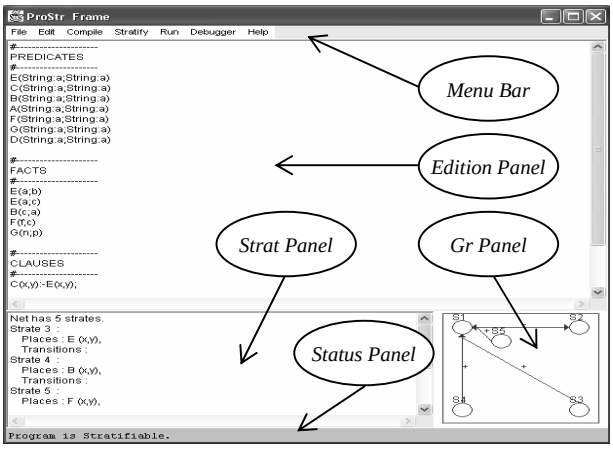

Fig. 4. STRPRO interface

STRPRO is composed of eight main modules:

1. *Editor module*: *STRPRO* offers two different ways for the edition of a normal program: textual and graphical. In addition, users can save and reload their programs into and from files.
2. *Compiler module*: the edition of normal programs in textual mode should be followed by compilation. This operation consists on the syntax analysis of the given normal program and its mapping - in case of validity - into its corresponding SEPN.
3. *Stratifiability Checker module*: *STRPRO* contains a module that checks the stratifiability property of a given program. This operation consists on the verification of absence of recursion through negation in the SEPN.
4. *Maximal Stratification Extractor module*: Once the normal program is stratifiable, user can call the "*Stratify*" command in order to extract the reduced graph *Gr*.
5. *Standard Model Computation module*: After the stratification of the program, we can compute its standard model.
6. *Query Evaluation module*: Since the standard model corresponds to net marking, query evaluation in *STRPRO* is simply the token existence test.
7. *Update Operations module*: After each update operation, the tool checks the program stratifiability. In case of validity, the new maximal stratification is determined. After that, the standard model is updated.
8. *The Debugger module*: *STRPRO* provides users with the ability to debug their programs by the means of the *Debugger* (Fig. 5). The *Debugger* shows a graphical interface that presents the internal modelling of a given program (its corresponding SEPN). Using this interface, we can perform all already cited operations.

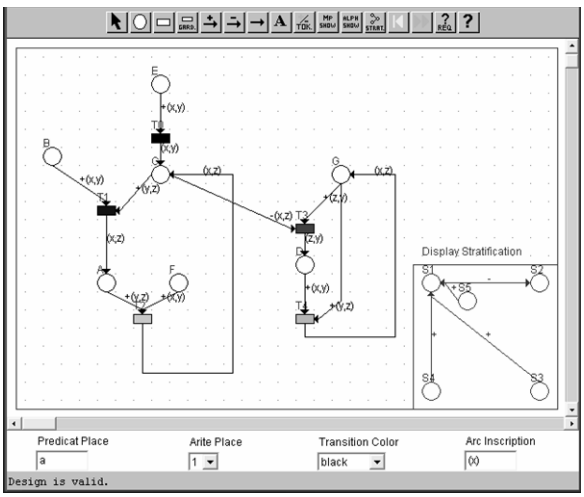

Fig. 5. STRPRO Debugger

VI. Conclusion

Stratification of normal programs received attention on behalf of researchers in the fields of the artificial intelligence and the deductive databases. Unfortunately, implementation aspects, in particular, representation structures and manipulation algorithms of stratified programs, were completely neglected. We proposed in this paper a complete implementation of stratification for the definition and manipulation of stratified programs. This implementation was validated with STRPRO tool. Proposed algorithms benefit from known algorithms in graph theory with an adaptation of some of them to SEPN context.

As we know, this tool is the first of its kind. It constitutes a complete and convivial compiler for stratified programs. Obtained results are encouraging for programs of nearly thirty clauses. We think that this tool helps understanding basic concepts of stratification. It can be used as a pedagogic tool with logic programming courses, and be used as a kernel component in expert systems and deductive databases.

In the future, we plan to:

1. Study the algorithms complexity in case of programs with significant clauses number (hundreds or more).
2. Extend STRPRO to support object paradigm and fuzzy logic.

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. F. 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] A. Grissa-Touzi, C. Jerad and H. Ounelli, "New approach for Manipulation of Stratified Programs", 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] C. Froidevaux, Mr. C Gaudel and M. Soria, '" Types of Data and Algorithmes' ', Ediscience International, 1990.

[12] R. S. Brink, "A Petri Net Design, Simulation, and Verification Tool", http://www.csh.rit.edu/~rick, Rochester Institute of Technology, Rochester, New York, 1996.