Fages' Theorem for Programs with Nested Expressions

Esra Erdem and Vladimir Lifschitz
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA
Email: {esra,vl}@cs.utexas.edu

Abstract

We extend a theorem by François Fages about the relationship between the completion semantics and the answer set semantics of logic programs to a class of programs with nested expressions permitted in the bodies of rules. Fages' theorem is important from the perspective of answer set programming: whenever the two semantics are equivalent, answer sets can be computed by propositional solvers, such as SATO, instead of answer set solvers, such as SMODELS. The need to extend Fages' theorem to programs with nested expressions is related to the use of choice rules in the input language of SMODELS.

1 Introduction

This note continues the line of research on the relationship between two theories of negation as failure—one based on program completion [Clark, 1978], the other based on stable models, or answer sets [Gelfond and Lifschitz, 1988]. A syntactic condition that guarantees the equivalence of these two concepts was given by Fages [1994]. Babovich, Erdem and Lifschitz [2000] generalized Fages' theorem and showed that results of this type can be applied to computing answer sets for a logic program: whenever the two semantics are equivalent, answer sets can be computed by propositional solvers, such as SATO [Zhang, 1997] or RELSAT [Bayardo and Schrag, 1997], instead of answer set solvers, such as SMODELS [Niemelä and Simons, 1996] and DLV [Eiter *et al.*, 1997].

This possibility is important from the perspective of answer set programming. The idea of this programming method is to reduce a given computational problem to computing an answer set for a logic program. Examples and references can be found in [Lifschitz, 1999].

Lloyd and Topor [1984] generalized the completion semantics to programs containing nested expressions (formulas) in the bodies of rules. A similar generalization of the answer set semantics was proposed in [Lifschitz *et al.*, 1999]. In this note we show how Fages' theorem on the relationship between completion and answer sets can be extended to these more general programs. We argue that this extension of Fages' theorem is important in connection with some of the syntactic constructs available in the input language of system SMODELS.

Consider a simple example. Program

$$\begin{array}{l} p \leftarrow not\ not\ p, \\ p \leftarrow p, q \end{array} \tag{1}$$

contains nested occurrences of negation as failure in the body of the first rule.[1] It belongs to the syntactic class for which our theorem guarantees the equivalence of the answer set semantics to the completion semantics. This program has two answer sets $\emptyset$, $\{p\}$; they are identical to the models of the completion of (1):

$$\begin{array}{l} p \equiv \neg\neg p \vee (p \wedge q), \\ q \equiv \bot. \end{array} \tag{2}$$

In the syntax of SMODELS, the first of rules (1) can be written as a "choice rule" $\{p\}$. The relationship between rules like this and nested expressions is systematically studied in [Ferraris and Lifschitz, 2001].

In the next section, we discuss a more interesting example of this kind—a formalization of the 8-queens problem in the language of SMODELS. Section 3 is a review of answer sets and completion for programs with nested expressions. After extending Fages' syntactic condition—"tightness"—to programs of this kind, we introduce our generalization of Fages' theorem (Section 4). The proof is expressed in terms of a generalization of the model-theoretic counterpart of completion introduced in [Apt *et al.*, 1988], called supportedness (Sections 5, 6).

2 The 8-Queens Problem

In the 8-queens problem, the goal is to find a configuration of 8 queens on an 8×8 chessboard such that no queen can be taken by any other queen. In other words, no two queens may be on the same row, on the same column, or on the same diagonal.

[1] The double negation in the first rule of (1) is redundant from the point of view of the completion semantics, but it does affect the program's answer sets. On the other hand, the second rule is redundant from the point of view of the answer set semantics, but, generally, dropping a rule like this can change a program's completion in an essential way.

```
row(1..8).
column(1..8).

1{occupied(R,C):row(R)}1 :- column(C).

:- occupied(R,C), occupied(R,C1),
   row(R), column(C), column(C1), C < C1.

:- occupied(R,C), occupied(R1,C1),
   row(R), column(C), row(R1), column(C1),
   C < C1, abs(R - R1) == abs(C - C1).
```

Figure 1: The 8-queens problem presented to SMODELS

The 8-queens problem can be presented to SMODELS as in Figure 1. The choice rule that follows the definitions of *row* and *column* instructs SMODELS to select atoms of the form $occupied(R, C)$ for including in an answer set in such a way that, for every column C, exactly one atom $occupied(R, C)$ be selected. The program has 92 answer sets, corresponding to all possible arrangements of 8 queens. Given this input file, SMODELS produces one of the solutions:

```
Stable Model: occupied(4,1) occupied(2,2) occupied(7,3)
occupied(5,4) occupied(1,5) occupied(8,6) occupied(6,7)
occupied(3,8)
```

Consider the grounded version of the program in Figure 1, with the domain predicates *row* and *column* dropped from the rules. It consists of the rules

$$1\{occupied(1, C), \ldots, occupied(8, C)\}1 \tag{3}$$

for all C in $\{1, \ldots, 8\}$,

$$\leftarrow occupied(R, C), occupied(R, C1) \tag{4}$$

for all $R, C, C1$ in $\{1, \ldots, 8\}$ such that $C < C1$, and

$$\leftarrow occupied(R, C), occupied(R1, C1) \tag{5}$$

for all $R, R1, C, C1$ in $\{1, \ldots, 8\}$ such that $C < C1$ and $|R - R1| = |C - C1|$. Rewritten in terms of nested expressions, as described in [Ferraris and Lifschitz, 2001, Section 6], rule (3) becomes

$$\begin{array}{l} occupied(R, C) \leftarrow not\ not\ occupied(R, C) \\ \leftarrow occupied(R, C), occupied(R1, C) \qquad (R < R1) \\ \leftarrow not\ occupied(1, C), \ldots, not\ occupied(8, C). \end{array} \tag{6}$$

The first of these rules allows each of the atoms *occupied*(R, C) to be included or not included in an answer set arbitrarily. The second rule prohibits the selections that include more than one atom *occupied*(R, C) with the same value of C. The third rule prohibits the selections that include no such atoms for some value of C.

To sum up, the program from Figure 1 can be rewritten as the union of programs (6), (4) and (5).

The results of this paper show that the answer sets for the program consisting of these rules can be equivalently described as the models of the program's completion. Consequently, an answer set for this program can be found by the Causal Calculator (CCALC)[2] — a system that can compute the completion of a program, even if the program contains nested expressions, and then can call a propositional solver to find a model of the completion.

We present the union of programs (6), (4) and (5) to CCALC as shown in Figure 2.[3] CCALC produces the following output, using SATO to find a model:

```
Satisfying Interpretation:  occupied(1,5)  occupied(2,8)
occupied(3,4) occupied(4,1)  occupied(5,3)  occupied(6,6)
occupied(7,2)  occupied(8,7)
```

In our experiments, SMODELS took 0.06 seconds to find a solution to the 8-queens problem, and SATO took 0.01 seconds. For 20 queens, the run time of SMODELS was 200 seconds, and the run time of SATO was 0.08 seconds. For 30 queens, SMODELS did not terminate after many hours of search; the run time of SATO was only 0.29 seconds.[4] This example shows that, given a representation of a computational problem in the input language of SMODELS, it is sometimes faster to find a solution by running a propositional solver on the completion of the corresponding program with nested expressions than by using SMODELS itself.

---

[2] `http://www.cs.utexas.edu/users/tag/cc/` .

[3] In CCALC, sorts of variables correspond to domain predicates of SMODELS. The symbol `cwAtomicFormula` in the declaration of `occupied` means "an atomic formula satisfying the closed world assumption".

[4] We used SMODELS 2.26 with LPARSE 0.99.59, and CCALC 1.9 with SATO 3.2. The times do not include the preprocessing done by LPARSE for SMODELS and by CCALC for SATO. The constraint logic programming system CLP [van Hentenryck, 1989] is computationally even more efficient in application to the n-queens problem: it takes 0.01 seconds to find a solution for 20 queens. According to [Pelov *et al.*, 2000], this problem can be also solved quickly using the abductive logic programming system SLDNFAC [Denecker and Van Nuffelen, 1999].

```
:- sorts
        row; column.

:- variables
        R,R1 :: row;
        C,C1 :: column.

:- constants
        1..8 :: row;
        1..8 :: column;
        occupied(row,column) :: cwAtomicFormula.

occupied(R,C) :- not (not occupied(R,C)).

:- occupied(R,C),occupied(R1,C), R < R1.

:- (/\R: -occupied(R,C)).

:- occupied(R,C), occupied(R,C1), C < C1.

:- occupied(R,C), occupied(R1,C1),
   C < C1, abs(R-R1) =:= abs(C-C1).
```

Figure 2: The 8-queens problem presented to CCALC

3 Programs

The words *atom* and *literal* are understood here as in propositional logic; we call the sign $\neg$ in a negative literal $\neg A$ *classical negation* to distinguish from the symbol for negation as failure (*not*). *Elementary formulas* are literals and the 0-place connectives $\bot$ and $\top$. *Formulas* are built from elementary formulas using the unary connective *not* and the binary connectives , (conjunction) and ; (disjunction). A *(nondisjunctive) rule* is an expression of the form

$$Head \leftarrow Body \tag{7}$$

where *Head* is a literal or $\bot$, and *Body* is a formula.[5] If $Head = \bot$, we will sometimes drop $\bot$ from the head; a rule with $Head = \bot$ is called a *constraint*.

A *(nondisjunctive) program* is a set of rules.

[5] In [Lifschitz *et al.*, 1999], the syntax of rules is more general: the head may be an arbitrary formula, in particular a disjunction.

We define when a consistent set X of literals *satisfies* a formula F (symbolically, $X \models F$) recursively, as follows:

- for elementary F, $X \models F$ if $F \in X$ or $F = \top$,
- $X \models not\ F$ if $X \not\models F$,
- $X \models (F, G)$ if $X \models F$ and $X \models G$,
- $X \models (F; G)$ if $X \models F$ or $X \models G$.

A consistent set X of literals is *closed under* Π if, for every rule (7) in Π, $Head \in X$ whenever $X \models Body$.

Let Π be a program without negation as failure. We say that X is an *answer set* for Π if X is minimal among the consistent sets of literals closed under Π. It is easy to see that there can be at most one such set. For instance, the answer set for the program

$$\begin{array}{l} p \leftarrow \top \\ p \leftarrow p, q \end{array} \tag{8}$$

is $\{p\}$.

The *reduct* Π^X of a program Π relative to a set X of literals is obtained from Π by replacing every maximal occurrence of a formula of the form $not\ F$ in Π (that is, every occurrence of $not\ F$ that is not in the range of another not) with $\bot$ if $X \models G$, and with $\top$ otherwise.[6] A consistent set X of literals is an *answer set* for Π if it is the answer set for the reduct Π^X. For instance, $\{p\}$ is an answer set for (1) since it is an answer set for the reduct (8) of (1) relative to $\{p\}$.

We say that a formula (or a program) is *normal* if it does not contain classical negation. A normal formula F can be identified with the propositional formula obtained from F by replacing every comma with $\wedge$, every semicolon with $\vee$, and not with $\neg$.

Consider a finite normal program Π. The completion of Π is defined as follows. If A is an atom or the symbol $\bot$, by $Comp(\Pi, A)$ we denote the propositional formula

$$A \equiv (Body_1 \vee \ldots \vee Body_k) \tag{9}$$

where the disjunction extends over all rules

$$A \leftarrow Body_i \tag{10}$$

[6] This definition is equivalent to the recursive definition of the reduct given in [Lifschitz *et al.*, 1999].

in Π with the head A.[7] The *completion* of Π is the set of formulas $Comp(\Pi, A)$ for all A. For instance, the bodies of rules (1), written as propositional formulas are $\neg\neg p$ and $p \wedge q$; for this program Π, formulas (2) are $Comp(\Pi, p)$ and $Comp(\Pi, q)$. In addition to these two formulas, the completion of this program includes also $Comp(\Pi, \bot)$, which is the tautology $\bot \equiv \bot$.

As in the case of normal programs without nesting, an answer set for any normal program Π satisfies the completion of Π. In the next section we will see under what conditions the converse is true.

4 A Generalization of Fages' Theorem

Although the theorem below applies to normal programs only, it is convenient to define tightness for programs that may contain classical negation.

A formula is called *negative* if every occurrence of an atom in this formula is in the scope of a negation as failure.

We consider nondisjunctive programs consisting of rules of the form

$$Head \leftarrow P, N \tag{11}$$

where P is an arbitrary formula, and N is a negative formula. In applications, P will be usually a formula that does not contain negation as failure. Any nondisjunctive program can be turned into a program of this kind using the "equivalent transformations" discussed in [Lifschitz *et al.*, 1999, Section 4]. For instance, the rule

$$p \leftarrow q; not\ r$$

does not have the form (11), but it can be equivalently replaced by the pair of rules

$$\begin{array}{l} p \leftarrow q, \top \\ p \leftarrow \top, not\ r. \end{array} \tag{12}$$

In fact, some disjunctive programs can be converted to this special form as well; for instance,

$$p; not\ q \leftarrow r$$

can be equivalently replaced by

$$p \leftarrow r, not\ not\ q$$

([Lifschitz *et al.*, 1999], Proposition 6(iii)).

[7] This is essentially the definition from [Lloyd and Topor, 1984] restricted to the propositional case.

An occurrence of a formula F in a formula is *singular* if the symbol before this occurrence is $\neg$; otherwise, the occurrence is *regular*. It is clear that the occurrence of F can be singular only if F is an atom. For any formula G, by $lit(G)$ we denote the set of all literals having regular occurrences in G. For instance, $lit(p; not\ \neg r) = \{p, \neg r\}$.

Now we are ready to show how Fages' syntactic condition is extended to programs with nested expressions.

A program Π whose rules have the form (11) is *tight* on a set X of literals if there exists a function λ from X to ordinals satisfying the following condition:

(*) for every rule (11) in Π, if $Head \in X$ and $X \models (P, N)$ then, for all $L \in X \cap lit(P)$, $\lambda(L) < \lambda(Head)$.

For instance, program (1) written as

$$\begin{array}{l} p \leftarrow \top, not\ not\ p, \\ p \leftarrow (p, q), \top \end{array}$$

is tight on $\{p\}$: take $\lambda(p) = 0$. Program (12) is tight on $\{p, q, r\}$: take $\lambda(p) = 1$, $\lambda(q) = \lambda(r) = 0$.

If the program is finite then, without loss of generality, the values of λ can be assumed to be finite.

For programs whose rules are of the form

$$Head \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$$

where each L_i is a literal, the definition of tightness above is slightly more general than the definition given in [Babovich *et al.*, 2000]. If we take P to be $L_1, \ldots, L_m$ and N to be $not\ L_{m+1}, \ldots, not\ L_n$ then the condition $X \models (P, N)$ can be written as

$$L_1, \ldots, L_m \in X,\ \ L_{m+1}, \ldots, L_n \notin X,$$

and it implies that $X \cap lit(P) = \{L_1, \ldots, L_m\}$. In [Babovich *et al.*, 2000], the restriction $L_{m+1}, \ldots, L_n \notin X$ is not included in the definition of tightness.[8] For instance, the program

$$\begin{array}{l} p \leftarrow \top, \top \\ q \leftarrow \top, \top \\ p \leftarrow p, not\ q \end{array}$$

[8] The possibility of including this restriction was suggested to us by Hudson Turner on May 2, 2001.

is tight on $\{p, q\}$ in the sense of (*), but it is not tight in the sense of [Babovich *et al.*, 2000]. Note that $\{p, q\}$ is both a model of the completion of this program and its answer set.

Our generalization of Fages' theorem is stated as follows:

Theorem 1 *For any finite normal program* Π *whose rules have the form (11) and any set* X *of atoms such that* Π *is tight on* X*,* X *is an answer set for* Π *iff* X *satisfies the completion of* Π.

For instance, program (1) is tight on the models $\emptyset$, $\{p\}$ of its completion (2). By the theorem above, it follows that $\emptyset$ and $\{p\}$ are the answer sets for (1).

In the modification of Theorem 1 stated below, the rules of Π are not required to have the form (11), and tightness is replaced by a simpler condition. We say that a literal L is a *positive body element* of a nondisjunctive program Π if Π contains a rule $Head \leftarrow Body$, with $Head \neq \bot$, such that $Body$ contains a regular occurrence of L that is not in the scope of negation as failure. We denote by $pos(\Pi)$ the set of positive body elements of Π. For instance, if Π is (1) then $pos(\Pi) = \{p, q\}$; if Π is (4)–(6) then $pos(\Pi) = \emptyset$.

Theorem 2 *For any finite normal program* Π *and any set* X *of atoms disjoint from* $pos(\Pi)$*,* X *is an answer set for* Π *iff* X *satisfies the completion of* Π.

For instance, this theorem shows that all models of the completion of the 8-queens program (4)–(6) are answer sets.

Theorems 1 and 2 are proved in the next two sections.

5 Supported Sets

The proof of Theorem 1 below is expressed in terms of a model-theoretic counterpart of completion—supportedness.

We say that a set X of literals is *supported by* a program Π if, for every literal $L \in X$, there exists a rule (7) in Π such that $Head = L$ and $X \models Body$. In application to finite normal programs, the combination of closure and supportedness exactly corresponds to the program's completion:

Proposition 1 *For any finite nondisjunctive normal program* Π*, a set of atoms satisfies the completion of* Π *iff it is closed under and supported by* Π.

Proof. Let Π be a finite nondisjunctive normal program. Recall that the completion of Π consists of equivalences (9) where A is an atom or the symbol $\bot$. It is clear that a set X satisfies the completion of Π iff, for each A,

(a) for every rule (10) in Π with the head A, if $X \models Body_i$ then $A \in X$, and

(b) if $A \in X$ then $X \models Body_i$ for some rule (10) in Π with the head A.

(Given the convention about writing normal formulas in the syntax of propositional logic introduced in Section 3, the relation $X \models F$ restricted to normal formulas F is equivalent to the satisfaction relation of classical logic.) Condition (a) expresses that X is closed under Π, and condition (b) expresses that X is supported by Π.

An answer set for a program Π is closed under and supported by Π:

Proposition 2 *For any nondisjunctive program Π and any consistent set X of literals, if X is an answer set for Π then X is closed under and supported by Π.*

Under the tightness condition, the converse holds as well—the sets closed under and supported by Π are answer sets for Π:

Proposition 3 *For any program Π whose rules have the form (11) and any consistent set X of literals such that Π is tight on X, if X is closed under and supported by Π then X is an answer set for Π.*

Propositions 2 and 3 generalize Theorem 1 to programs that may contain classical negation and may consist of infinitely many rules. The proofs of these propositions are given in the next section. Theorem 1 is an immediate consequence of Propositions 1–3.

6 Proofs

Lemma 1 *Given a formula F without negation as failure and two sets Z, Z' of literals such that $Z' \subseteq Z$, if $Z' \models F$ then $Z \models F$.*

Proof. Immediate by structural induction.

The following lemma is the special case of Proposition 2 in which Π is assumed to be a program without negation as failure.

Lemma 2 *For any nondisjunctive program Π without negation as failure and any consistent set X of literals, if X is an answer set for Π, then X is closed under and supported by Π.*

Proof. Let Π be a nondisjunctive program without negation as failure and X be an answer set for Π. By the definition of an answer set for programs without negation as failure, X is closed under Π. To prove supportedness, take any literal L in X. Since X is minimal among the sets closed under Π, $X \setminus \{L\}$ is not closed under Π. This means that Π contains a rule (7) such that $X \setminus \{L\} \models \mathit{Body}$ but $\mathit{Head} \notin X \setminus \{L\}$. By Lemma 1, $X \models \mathit{Body}$. Since X is closed under Π, it follows that $\mathit{Head} \in X$, so that $\mathit{Head} = L$.

The definition of the reduct F^X of a formula F is similar to the definition of the reduct of a program given in Section 3.

Lemma 3 *For any formula F, any nondisjunctive program Π, and any consistent set X of literals,*

(i) $X \models F$ *iff* $X \models F^X$*;*

(ii) X *is closed under* Π *iff* X *is closed under* Π^X*;*

(iii) X *is supported by* Π *iff* X *is supported by* Π^X*.*

Proof. Part (i) is immediate by structural induction; parts (ii) and (iii) follow from (i).

Proof of Proposition 2. Consider a nondisjunctive program Π and an answer set X for Π. By the definition of an answer set, X is an answer set for Π^X. Then, by Lemma 2, X is closed under and supported by Π^X. By Lemma 3(ii,iii), it follows that X is closed under and supported by Π.

Lemma 4 *For any formula F, and any set X of literals, $X \models F$ iff $X \cap \mathit{lit}(F) \models F$.*

Proof. Immediate by structural induction.

The following lemma is the special case of Proposition 3 in which Π is assumed to be a program without negation as failure.

Lemma 5 *Let Π be a program without negation as failure whose rules have the form (11). For any consistent set X of literals such that Π is tight on X, if X is closed under and supported by Π, then X is an answer set for Π.*

Proof. By the definition of an answer set for programs without negation as failure, we need to show that no proper subset of X is closed under Π. Let Y be a proper subset of X, and let λ be a function from X to ordinals satisfying condition (*) from the definition of tightness (Section 4). Take a literal $L \in X \setminus Y$ such that $\lambda(L)$ is minimal. Since X is supported by Π, there is a rule

$$L \leftarrow P, N$$

in Π such that

$$X \models (P, N). \tag{13}$$

By the choice of λ, for all $L' \in X \cap lit(P)$,

$$\lambda(L') < \lambda(L).$$

By the choice of L, no literal L' satisfying this inequality may belong to $X \setminus Y$, so that $X \cap lit(P)$ is disjoint from $X \setminus Y$. Consequently, $X \cap lit(P) \subseteq Y$. Since Π does not contain negation as failure, and N is a negative formula, $lit(N) = \emptyset$. Consequently, $lit(P, N) = lit(P)$, so that

$$X \cap lit(P, N) \subseteq Y. \tag{14}$$

By Lemma 4, we can conclude from (13) that

$$X \cap lit(P, N) \models (P, N).$$

In view of (14), it follows by Lemma 1 that $Y \models (P, N)$. Consequently, Y is not closed under Π.

Lemma 6 *For any program Π whose rules have the form (11) and any consistent set X of literals, if Π is tight on X then so is Π^X.*

Proof. Let Π be a program whose rules have the form (11) and X be a consistent set of literals. Suppose that Π is tight on X. Then there exists a function λ from X to ordinals such that, for every rule (11) in Π, if $Head \in X$ and $X \models (P, N)$ then, for all $L \in X \cap lit(P)$,

$$\lambda(L) < \lambda(Head).$$

By Lemma 3(i) and the definition of the reduct, $X \models (P, N)$ iff $X \models (P^X, N^X)$. By the definition of the reduct, $lit(P^X) \subseteq lit(P)$. Therefore, for every rule (11) in Π, if $Head \in X$ and $X \models (P^X, N^X)$ then, for all $L \in X \cap lit(P^X)$, the inequality above is satisfied. Since every rule of Π^X has the form

$$Head \leftarrow P^X, N^X$$

for some rule (11) in Π, it follows that Π^X is tight on X.

Proof of Proposition 3. Consider a program Π whose rules have the form (11) and a consistent set X of literals such that Π is tight on X. Assume that X is closed under and supported by Π. Then, by Lemma 3(ii,iii), X is closed under and supported by Π^X. By Lemma 6, since Π is tight on X, so is Π^X. Hence, by Lemma 5, X is an answer set for Π^X, and consequently an answer set for Π.

Proof of Theorem 2. Let Π be a finite normal program and let X be a set of literals disjoint from $lit(\Pi)$. As discussed in Section 4, the transformations from [Lifschitz *et al.*, 1999] allow us to turn Π into a program Π' with the same answer sets such that every rule of Π' has the form (11) where P does not contain negation as failure. The examination of these transformations shows that the completion of Π' is equivalent to the completion of Π. Furthermore, these transformations do not change the set of positive body elements, so that $pos(\Pi') = pos(\Pi)$. Consequently, X is disjoint from $pos(\Pi')$, which implies that X is disjoint from $lit(P)$ for every rule (11) of Π'. It follows that condition (*) from the definition of tightness (Section 4) applied to Π' holds trivially for any choice of λ. Then, by Theorem 1, X is an answer set for Π' iff X satisfies the completion of Π'. Consequently, this condition holds for Π as well.

7 Conclusion

We extended the theorem by François Fages that describes the relationship between the completion semantics and the answer set semantics of logic programs to programs with nested expressions permitted in the bodies of rules. The study of this relationship is important from the perspective of answer set programming, and nested expressions are interesting because of their relation to some syntactic constructs that play an important role in the input language of SMODELS. Experiments show that, given a representation of a computational problem in that language, it is sometimes faster to find a solution by running a propositional solver on the completion of the corresponding program with nested expressions than by using SMODELS.

Acknowledgments

We are grateful to Joohyung Lee and Hudson Turner for comments on a draft of this paper. This work was partially supported by National Science Foundation under grant IIS-9732744. The first author was also supported by a NATO science fellowship.

References

[Apt *et al.*, 1988] Krzysztof Apt, Howard Blair, and Adrian Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, San Mateo, CA, 1988.

[Babovich *et al.*, 2000] Yuliya Babovich, Esra Erdem, and Vladimir Lifschitz. Fages' theorem and answer set programming.[9] In *Proc. NMR-2000*, 2000.

[Bayardo and Schrag, 1997] Roberto Bayardo and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proc. IJCAI-97*, pages 203–208, 1997.

[Clark, 1978] Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[Denecker and Van Nuffelen, 1999] Marc Denecker and Bert Van Nuffelen. Experiments for integration CLP and abduction.[10] In Krysztof R. Apt, Antonios C. Kakas, Eric Monfroy, and Francesca Rossi, editors, *Proceedings of the 1999 ERCIM/COMPULOG workshop on Constraints*, pages 1–15, 1999.

[Eiter *et al.*, 1997] Thomas Eiter, Nicola Leone, Christinel Mateis, Gerald Pfeifer, and Francesco Scarcello. A deductive system for non-monotonic reasoning. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 363–374. Springer-Verlag, 1997.

[Fages, 1994] François Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.

[Ferraris and Lifschitz, 2001] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions.[11] Submitted for publication, 2001.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Logic Programming: Proc. Fifth Int'l Conf. and Symp.*, pages 1070–1080, 1988.

[9] `http://arxiv.org/abs/cs.ai/0003042` .

[10] `http://www.cs.kuleuven.ac.be/cgi-bin-dtai/publ_info.pl?id=20013` .

[11] `http://www.cs.utexas.edu/users/vl/mypapers/weight.ps` .

[Lifschitz *et al.*, 1999] Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.

[Lifschitz, 1999] Vladimir Lifschitz. Answer set planning. In *Proc. ICLP-99*, pages 23–37, 1999.

[Lloyd and Topor, 1984] John Lloyd and Rodney Topor. Making Prolog more expressive. *Journal of Logic Programming*, 3:225–240, 1984.

[Niemelä and Simons, 1996] Ilkka Niemelä and Patrik Simons. Efficient implementation of the well-founded and stable model semantics. In *Proc. Joint Int'l Conf. and Symp. on Logic Programming*, pages 289–303, 1996.

[Pelov *et al.*, 2000] Nikolay Pelov, Emmanuel De Mot, and Marc Denecker. Logic programming approaches for representing and solving constraint satisfaction problems : a comparison.[12] In Parigot M. and Voronkov A., editors, *Proceedings of the 7th International Conference on Logic for Programming and Automated Reasoning*, volume 1955 of *Lecture Notes in Artificial Intelligence*, pages 225–239. Springer, 2000.

[van Hentenryck, 1989] Pascal van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[Zhang, 1997] Hantao Zhang. SATO: An efficient propositional prover. In *Proc. CADE-97*, 1997.

[12] `http://www.cs.kuleuven.ac.be/cgi-bin-dtai/publ_info.pl?id=32258` .