Nested Expressions in Logic Programs

Vladimir Lifschitz and Lappoon R. Tang

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA

Hudson Turner

Department of Computer Science
University of Minnesota at Duluth
Duluth, MN 55812, USA

We extend the answer set semantics to a class of logic programs with nested expressions permitted in the bodies and heads of rules. These expressions are formed from literals using negation as failure, conjunction (,) and disjunction (;) that can be nested arbitrarily. Conditional expressions are introduced as abbreviations. The study of equivalent transformations of programs with nested expressions shows that any such program is equivalent to a set of disjunctive rules, possibly with negation as failure in the heads. The generalized answer set semantics is related to the Lloyd-Topor generalization of Clark's completion and to the logic of minimal belief and negation as failure.

1. Introduction

In most papers on the semantics of logic programming, the body of a rule is supposed to be a list of syntactically simple expressions, such as atoms or literals. The syntax of Prolog, however, permits nested expressions. Besides conjunctions (p, q) and negation as failure $(not\, p)$, these expressions may include, for instance, disjunctions $(p; q)$ and if-then-else constructs $(p \rightarrow q; r)$, nested arbitrarily.

The use of such expressions does not appear to make a program any less declarative. Declaratively, the rule

$$s \leftarrow (p \rightarrow q; r), t \tag{1}$$

has the same meaning as the pair of rules

$$\begin{array}{l} s \leftarrow p, q, t \\ s \leftarrow not\, p, r, t \end{array} \tag{2}$$

but it is more concise. The usefulness of the if-then-else operator is related also to the fact that p is evaluated only once when (1) is processed by Prolog; in (2), p may be evaluated twice. To eliminate this unnecessary recomputation without the use of the if-then-else construct, one would have to use cut. For these reasons, the use of nested expressions is standard practice in Prolog programming.

The first declarative semantics for logic programs with nested expressions was proposed by Lloyd and Topor [17], who extended the completion semantics [3] to programs with arbitrary first-order formulas in the bodies of rules. In this note, we propose a semantics for programs with nested expressions that is based on the answer set ("stable model") approach [7]. The role of logic programs under the answer set semantics as a knowledge representation language has been growing in recent years. This is evidenced by the development of software systems for computing answer sets, such as SMODELS [19] and DLV [5], by the emergence of "stable model programming" as an alternative logic programming paradigm [18] and by the use of answer sets for specifying planning problems [21,14] and for the development of efficient planning algorithms [4]. The extension of the answer set semantics proposed below can serve as a specification for extending systems like SMODELS and DLV to programs with nested expressions.

In this paper, we restrict attention to propositional programs; the semantics can be extended to programs with variables by grounding. Both the heads and bodies of the rules in such programs are formed from literals (atoms possibly preceded by the classical negation sign $\neg$) using the operators

$$, \quad ; \quad not \tag{3}$$

that can be nested arbitrarily. Conditional expressions are introduced as abbreviations.

An interesting fact about nested expressions under the answer set semantics is that two negation as failure operators in a row do not, generally, cancel each other. Our definition, applied, for instance, to the program

$$p \leftarrow not\; not\; p\,, \tag{4}$$

gives both $\emptyset$ and $\{p\}$ as the answer sets, although

$$p \leftarrow p$$

has only one answer set $\emptyset$. The following informal argument shows that this is in fact reasonable. In (4), we can "denote" the subformula *not* p by q, and (4) will turn into

$$\begin{aligned} p &\leftarrow \mathit{not}\, q\,, \\ q &\leftarrow \mathit{not}\, p\,. \end{aligned}$$

The answer sets for this program are $\{p\}$ and $\{q\}$. After dropping the "auxiliary" symbol q from each of them, we will get the two answer sets shown above.

In Sections 2 and 3 we define the syntax and semantics of programs with nested expressions. In Section 4, we turn to the study of equivalent transformations of programs with nested expressions,[1] and show that any such program can be equivalently transformed into a set of "disjunctive rules" in the sense of Section 5.1 of [13]—expressions of the form

$$\begin{aligned} L_1; \dots; L_k; \mathit{not}\, L_{k+1}; \dots; \mathit{not}\, L_l \leftarrow & \\ L_{l+1}, \dots, L_m, \mathit{not}\, L_{m+1}, \dots, \mathit{not}\, L_n & \end{aligned} \tag{5}$$

where $0 \leq k \leq l \leq m \leq n$ and all L_i are literals.[2] A somewhat unorthodox feature of (5) is the possibility of negation as failure in the head. This possibility, useful in abductive logic programming [10] and in the theory of updates [1], is essential here also, because in the process of "unwinding" a nested expression in the body some occurrences of negation as failure can move to the head.

In Section 5 we discuss the relation of our proposal to the Lloyd and Topor system mentioned above. Several authors, including Lin and Shoham [16], Lifschitz [12], Herre and Wagner [8] and Pearce [20], described ways to embed logic programs under the answer set semantics into languages that are closed under the formation rules of propositional logic. In Section 6 the embedding due to Lifschitz is extended to logic programs with nested expressions.

Proofs of theorems are given in Section 7.

[1] These transformations are somewhat similar to the "algebraic laws" of Prolog discussed in [9] on the basis of a procedural view of logic programming.
[2] In [13], the vertical bar is used instead of the semicolon to separate literals in the head of a rule.

2. Syntax

The words *atom* and *literal* are understood here as in propositional logic. *Elementary formulas* are literals and the 0-place connectives $\bot$ ("false") and $\top$ ("true"). *Formulas* are built from elementary formulas using the unary connective *not* and the binary connectives , (conjunction) and ; (disjunction). A *rule* is an expression of the form

$$F \leftarrow G$$

where F and G are formulas, called the *head* and the *body* of the rule.

For any formulas F, G and H,

$$F \rightarrow G; H$$

stands for the formula

$$(F, G); (not\, F, H)\,.$$

We will write a rule of the form $F \leftarrow \top$ as $F \leftarrow$ and identify it with formula F. Rules of the form $\bot \leftarrow G$ will be called *constraints* and written as $\leftarrow G$. Given these conventions, the definition of a rule given above is a generalization of rules in the sense of [7], and even of "disjunctive rules" (5).

A *program* is a set of rules.

Note that, syntactically, the status of the classical negation operator $\neg$ is different from the status of operators (3): classical negation is allowed only in front of an atom. About an occurrence of a formula F in a formula or a rule we will say that it is *singular* if the symbol before this occurrence is $\neg$; otherwise the occurrence is *regular*. It is clear that an occurrence of F can be singular only if F is an atom. For instance, in the formula

$$not\, p, not\, \neg p \tag{6}$$

the first occurrence of p is regular, and the second is singular. This difference is important because the result of replacing a regular occurrence of F in a formula or a rule by another formula G is again a formula or a rule, but for a singular occurrence this is not necessarily the case. For instance, the result of replacing the first occurrence of p in (6) by (q, r) is a formula, but the result of applying the same operation to the second occurrence is not.

3. Semantics

Recall that the definition of an answer set [7] consists of two parts. First this concept is defined for the "basic" case of programs that do not contain negation as failure. Then the "reduct" of a program relative to a set of literals is defined, and this is used to reduce the general case to the basic case. The extension of this definition to programs with nesting follows the same plan.

The formulas, rules and programs that do not contain the negation as failure operator *not* will be called *basic*. We define when a consistent set X of literals *satisfies* a basic formula F (symbolically, $X \models F$) recursively, as follows:

- for elementary F, $X \models F$ if $F \in X$ or $F = \top$.
- $X \models (F, G)$ if $X \models F$ and $X \models G$.
- $X \models (F; G)$ if $X \models F$ or $X \models G$.

Let Π be a basic program. A consistent set X of literals is *closed* under Π if, for every rule $F \leftarrow G$ in Π, $X \models F$ whenever $X \models G$. We say that X is an *answer set* for Π if X is minimal among the consistent sets of literals closed under Π.

Example 1. Consider the program whose only rule is

$$q \leftarrow p; \neg p\,. \tag{7}$$

Being closed under (7) is characterized by the following condition: if $p \in X$ or $\neg p \in X$ then $q \in X$. It is clear that the only answer set for (7) is empty. If we add the rule p (that is, $p \leftarrow \top$) to this program then $\{p, q\}$ will be the only answer set.

The *reduct* of a formula, rule or program relative to a consistent set X of literals is defined recursively, as follows:

- for elementary F, $F^X = F$.
- $(F, G)^X = (F^X, G^X)$.
- $(F; G)^X = (F^X; G^X)$.
- $(not\ F)^X = \begin{cases} \bot, & \text{if } X \models F^X, \\ \top, & \text{otherwise.} \end{cases}$
- $(F \leftarrow G)^X = F^X \leftarrow G^X$.
- $\Pi^X = \{(F \leftarrow G)^X : F \leftarrow G \in \Pi\}$.

A consistent set X of literals is an *answer set* for Π if it is an answer set for the reduct Π^X.

Example 2. Consider the program whose only rule is

$$p \leftarrow (q \rightarrow r; not\, s)\,, \tag{8}$$

that is,

$$p \leftarrow (q, r); (not\, q, not\, s)\,.$$

Take $X = \{p\}$; let us check that X is an answer set for this program. Note that $q^X = q$, so that $q^X \notin X$ and $X \not\models q^X$. Consequently $(not\, q)^X = \top$. Similarly, $(not\, s)^X = \top$. It follows that

$$((q, r); (not\, q, not\, s))^X = (q, r); (\top, \top)\,.$$

Consequently the reduct of (8) relative to X is

$$p \leftarrow (q, r); (\top, \top)\,.$$

The only answer set for this program is $\{p\}$, so X is indeed an answer set for (8). (There are no other answer sets.)

Example 3. Consider program (4). For $X_1 = \emptyset$, $p^{X_1} \notin X_1$, which means that $X_1 \not\models p^{X_1}$ and consequently $(not\, p)^{X_1} = \top$. It follows that $X_1 \models (not\, p)^{X_1}$, so that $(not\, not\, p)^{X_1} = \bot$. Hence the reduct of (4) relative to X_1 is

$$p \leftarrow \bot\,.$$

The only answer set for this program is empty, so X_1 is indeed an answer set for (4). Now take $X_2 = \{p\}$. Note that $p^{X_2} \in X_2$, so $X_2 \models p^{X_2}$. Consequently $(not\, p)^{X_2} = \bot$. It follows that $X_2 \not\models (not\, p)^{X_2}$, so that $(not\, not\, p)^{X_2} = \top$. Hence the reduct of (4) relative to X_2 is

$$p \leftarrow \top\,.$$

The only answer set for this program is $\{p\}$, so X_2 is an answer set for (4) also.

Proposition 1. For a program whose rules have form (5), the answer sets according to the definition of an answer set given above are exactly the consistent answer sets according to the definition from [13].

Unlike the definitions of an answer set given in [7,15,13], the definition introduced above does not allow for an inconsistent answer set. All three previous definitions agree, where they overlap, on the question of consistent answer sets, but there is some disagreement among them on inconsistent answer sets. For instance, according to [15], the program

$$\mathit{not}\, p$$

has two answer sets: $\emptyset$ and the set of all literals. According to [13], it has only the empty answer set. (In [7], negation as failure in the head is not considered.) As another example, consider the program

$$\begin{array}{l} p\,, \\ \leftarrow p\,. \end{array}$$

According to [7] and [13], this program has no answer sets. In [15], it has a single answer set: the set of all literals.

Let Π be a set of constraints. A consistent set X of literals *violates* Π if it is not closed under Π^X, that is to say, if Π includes a constraint $\leftarrow G$ such that $X \models G^X$. For instance, X violates $\{\leftarrow p\}$ if $p \in X$; X violates $\{\leftarrow \mathit{not}\, p\}$ if $p \notin X$. The effect of adding a set of constraints to a program is to rule out the answer sets that violate these constraints. More precisely:

Proposition 2. Let Π_1, Π_2 be programs such that every rule in Π_2 is a constraint. A consistent set of literals is an answer set for $\Pi_1 \cup \Pi_2$ iff it is an answer set for Π_1 and does not violate Π_2.

4. Equivalent Transformations

In this section we describe several equivalent transformations of programs with nested expressions. These transformations can turn any program into an equivalent program whose rules have form (5).

Equivalence can be understood here as a condition stronger than merely having the same answer sets. About programs Π_1 and Π_2 we say that they are *equivalent* if, for any consistent sets X and Y of literals, X is closed under Π_1^Y iff X is closed under Π_2^Y. It is clear that the reducts of two equivalent programs relative to the same set of literals have the same answer sets. Consequently, any two equivalent programs have the same answer sets.

It is clear also that if Π_1 is equivalent to Π_2 then, for any program Π, $\Pi_1 \cup \Pi$ is equivalent to $\Pi_2 \cup \Pi$.

Some of the transformations discussed in this section replace subformulas in the program by other formulas. Some transformations move subformulas from the body of a rule to its head, or from the head to the body. We will also consider transformations that break a rule into two rules.

In connection with transformations of the first kind, we need the following definition. A formula F is *equivalent* to a formula G (symbolically, $F \Leftrightarrow G$) if, for any consistent sets X and Y of literals, $X \models F^Y$ iff $X \models G^Y$. Here is why this definition is of interest to us:

Proposition 3. Let Π be a program, and let F, G be a pair of equivalent formulas. Any program obtained from Π by replacing some regular occurrences of F by G is equivalent to Π.

Here is a collection of useful facts about the equivalence of formulas:

Proposition 4. For any formulas F, G, H,

(i) $F, G \Leftrightarrow G, F$ and $F; G \Leftrightarrow G; F$.

(ii) $(F, G), H \Leftrightarrow F, (G, H)$ and $(F; G); H \Leftrightarrow F; (G; H)$.

(iii) $F, (G; H) \Leftrightarrow (F, G); (F, H)$ and $F; (G, H) \Leftrightarrow (F; G), (F; H)$.

(iv) $not\,(F, G) \Leftrightarrow not\,F; not\,G$ and $not\,(F; G) \Leftrightarrow not\,F, not\,G$.

(v) $not\ not\ not\ F \Leftrightarrow not\ F$.

(vi) $F, \top \Leftrightarrow F$ and $F; \top \Leftrightarrow \top$.

(vii) $F, \bot \Leftrightarrow \bot$ and $F; \bot \Leftrightarrow F$.

(viii) if p is an atom then $p, \neg p \Leftrightarrow \bot$ and $not\,p; not\,\neg p \Leftrightarrow \top$.

(ix) $not\,\top \Leftrightarrow \bot$ and $not\,\bot \Leftrightarrow \top$.

Using Propositions 3 and 4, we can show that any formula can be converted to "disjunctive" and "conjunctive" normal forms, described in Proposition 5 below. A *simple conjunction* is a formula of the form

$$L_1, \ldots, L_k, not\ L_{k+1}, \ldots, not\ L_m, not\ not\ L_{m+1}, \ldots, not\ not\ L_n \tag{9}$$

and a *simple disjunction* is a formula of the form

$$L_1; \dots ; L_k; \mathit{not}\ L_{k+1}; \dots ; \mathit{not}\ L_m; \mathit{not}\ \mathit{not}\ L_{m+1}; \dots ; \mathit{not}\ \mathit{not}\ L_n \tag{10}$$

where $0 \leq k \leq m \leq n$, $n > 0$ and all L_i are literals.

Proposition 5. Any formula is equivalent to

(i) a formula of the form $F_1; \dots ; F_n$ where $n \geq 1$ and each F_i is a simple conjunction, and

(ii) a formula of the form $F_1, \dots , F_n$ where $n \geq 1$ and each F_i is a simple disjunction.

The following proposition describes further equivalent transformations of programs.

Proposition 6.

(i) $F, G \leftarrow H$ is equivalent to

$$F \leftarrow H\,,$$
$$G \leftarrow H\,.$$

(ii) $F \leftarrow G; H$ is equivalent to

$$F \leftarrow G\,,$$
$$F \leftarrow H\,.$$

(iii) $F \leftarrow G, \mathit{not}\ \mathit{not}\ H$ is equivalent to $F; \mathit{not}\ H \leftarrow G\,.$

(iv) $F; \mathit{not}\ \mathit{not}\ G \leftarrow H$ is equivalent to $F \leftarrow \mathit{not}\ G, H\,.$

From the facts stated above, we can derive:

Proposition 7. Any program is equivalent to a set of rules of form (5).

Examination of the proof of Proposition 7 shows that the rules of form (5) generated from a program Π do not contain negation as failure in the heads ($k = l$) if negation as failure in Π

(i) does not occur in the heads of rules, and

(ii) is not nested, that is, not applied to formulas containing negation as failure.

The answer sets for any collection of rules of form (5) without negation as failure in the heads are known to have the anti-chain property: an answer set for such a program cannot be a proper subset of another answer set for the same program. It follows that the answer sets for any collection of rules satisfying conditions (i) and (ii) form an anti-chain. Program (4) does not satisfy condition (ii), and one of its two answer sets is a subset of the other.

5. Relation to the Lloyd-Topor Semantics

The review of the Lloyd-Topor semantics below differs from its description in [17] in two ways. First, it is restricted to the propositional case. Second, it is stated in terms of "supported models" [2] instead of completion, which allows us to extend the theory to infinite programs. For finite programs, the two definitions are equivalent.

A *(propositional) Lloyd-Topor rule* is an expression of the form

$$p \leftarrow F$$

where p is an atom and F is a propositional formula. A *Lloyd-Topor program* is a set of Lloyd-Topor rules.

Recall that a *(propositional) interpretation* is a function from atoms to truth values. We will identify an interpetation with the set of atoms to which it assigns the value *true*.

Let Π be a Lloyd-Topor program. A *model* of Π is an interpretation I such that, for any rule $p \leftarrow F$ in Π such that I satisfies F, I satisfies p also. A model I of Π is *supported* if, for any atom p such that I satisfies p, there is a rule $p \leftarrow F$ in Π such that I satisfies F.

For instance, the Lloyd-Topor program whose only rule is $p \leftarrow p$ has two supported models: $\emptyset$ and $\{p\}$.

The translation τ of propositional formulas, and of Lloyd-Topor rules and programs, to our language is defined as follows, assuming that all connectives other than $\wedge$, $\vee$ and $\neg$ have been eliminated:

- for atomic F, $\tau F = \mathit{not\ not}\ F$.
- $\tau(F \wedge G) = \tau F, \tau G$.
- $\tau(F \vee G) = \tau F; \tau G$.
- $\tau(\neg F) = \mathit{not}\ \tau F$.

- $\tau(p \leftarrow F) = p \leftarrow \tau F$.
- $\tau\Pi = \{\tau(p \leftarrow F) : p \leftarrow F \in \Pi\}$.

The soundness of this translation is expressed by the following theorem:

Proposition 8. For any Lloyd-Topor program Π and any interpretation I, I is a supported model of Π iff I is an answer set for $\tau\Pi$.

Since the heads of the rules of $\tau\Pi$ are atoms, any answer set for this program is a set of atoms, that is to say, an interpretation. Consequently, we can conclude from Proposition 8 that every answer set for $\tau\Pi$ is a supported model of Π.

For instance, $\tau(p \leftarrow p)$ is (4). In accordance with Proposition 8, the two answer sets for (4) computed in Example 3 are identical to the two supported models of $p \leftarrow p$. As another example, the result of applying τ to

$$p \leftarrow \neg q \tag{11}$$

is

$$p \leftarrow not\ not\ not\ q\,.$$

According to Propositions 3 and 4(v), this program is equivalent to

$$p \leftarrow not\, q\,.$$

We see that, in application to (11), the Lloyd-Topor semantics is equivalent to the answer set semantics, modulo replacing $\neg$ by *not*. This is not surprising in view of the fact that the result of this replacement is a tight program (see [13], and the result from [6] reproduced there as Proposition 3.5).

Proposition 8 is similar to Theorem 5.10 from [11], which is limited to programs without nested expressions.

6. Relation to MBNF

In the propositional fragment of the logic of minimal belief and negation as failure (MBNF) [12], formulas are built from atoms using propositional connectives and two modal operators: B ("minimal belief") and *not*. An *MBNF theory* is a set of formulas (axioms) in this language. We will introduce a translation μ that maps formulas and rules to formulas of MBNF, and maps programs to MBNF theories. The translation is defined recursively, as follows:

- for elementary F, $\mu F = \mathsf{B}F$.
- $\mu(F,G) = \mu F \wedge \mu G$.
- $\mu(F;G) = \mu F \vee \mu G$.
- for elementary F, $\mu(\mathit{not}\ F) = \mathit{not}\ F$.
- $\mu(\mathit{not}\ (F,G)) = \mu(\mathit{not}\ F) \vee \mu(\mathit{not}\ G)$.
- $\mu(\mathit{not}\ (F;G)) = \mu(\mathit{not}\ F) \wedge \mu(\mathit{not}\ G)$.
- $\mu(\mathit{not}\ \mathit{not}\ F) = \neg\mu(\mathit{not}\ F)$.
- $\mu(F \leftarrow G) = \mu G \supset \mu F$.
- $\mu\Pi = \{\mu(F \leftarrow G) : F \leftarrow G \in \Pi\}$.

As an example, note that, in application to (4), μ gives the formula

$$\neg \mathit{not}\ p \supset \mathsf{B}p\,.$$

This translation is essentially an extension of the translation from Section 5 of [12] to programs with nested expressions. The translation $\mu\Pi$ of any program Π is an MBNF theory of the special type studied in [15]—a "theory with protected literals." This means that every occurrence of an atom in an axiom of this theory is a part of an expression of the form $\mathsf{B}L$ or $\mathit{not}\ L$, where L is a literal.

Recall that *models* of a propositional MBNF theory are defined as the pairs (I, S), where I is an interpretation and S is a set of interpretations, that satisfy a certain fixpoint condition (see [12], Section 4).

The following theorem establishes a correspondence between the answer sets for a program Π and the consistent models of the MBNF theory $\mu\Pi$ (that is, the models (I, S) with nonempty S). For any consistent set X of literals, by $\mathrm{Mod}(X)$ we denote the (nonempty) set of interpretations that satisfy all members of X.

Proposition 9. For any program Π, consistent set X of literals, and interpretation I, X is an answer set for Π iff $(I, \mathrm{Mod}(X))$ is a model of $\mu\Pi$. Moreover, every consistent model of $\mu\Pi$ can be represented in the form $(I, \mathrm{Mod}(X))$, where I is an interpretation and X is a consistent set of literals.

7. Proofs of Theorems

7.1. Proof of Proposition 1

Proposition 1. For a program whose rules have form (5), the answer sets according to the definition of an answer set given above are exactly the consistent answer sets according to the definition from [13].

The definition of an answer set in [13] differs from the one given above in three ways: the definition of closure under a basic program is different; the definition of the reduct is different; the inconsistent answer set is allowed. Proposition 1 follows from two lemmas:

Lemma 1. Let Π be a program whose rules have the form

$$L_1; \ldots ; L_k \leftarrow L_{k+1}, \ldots , L_m \tag{12}$$

where $0 \leq k \leq m$ and all L_i are literals, and let X be a consistent set of literals. X is closed under Π in the sense of this paper iff X is closed under Π in the sense of [13].

Lemma 2. Let Π be a program whose rules have the form (5), and let X, Y be consistent sets of literals. Y is closed under Π^X iff Y is closed under the reduct of Π relative to X in the sense of [13].

(According to the first lemma, there is no need to distinguish between the two meanings of "closed under" in the statement of the second lemma.)

Proof of Lemma 1. It is easy to verify that X is closed under Π in the sense of this paper iff, for every rule (12) in Π, X includes at least one of the literals $L_1, \ldots , L_k$ provided that X includes all of $L_{k+1}, \ldots , L_m$. This is exactly what it is for X to be closed under Π in the sense of [13]. $\square$

Proof of Lemma 2. For a program Π whose rules have form (5) and for a consistent set X of literals, Π^X can be characterized as the result of replacing each subformula of the form *not* L in Π by $\bot$ if $L \in X$, and by $\top$ otherwise. On the other hand, the reduct of Π relative to X in the sense of [13] is defined as the program obtained from Π by

- deleting every rule (5) such that at least one of $L_{k+1}, \ldots, L_l$ is not in X, or at least one of $L_{m+1}, \ldots, L_n$ is in X, and
- replacing each remaining rule (5) by

$$L_1; \ldots ; L_k \leftarrow L_{l+1}, \ldots, L_m \ .$$

It is easy to verify that this program can be obtained from Π^X by

- deleting every rule (5) such that its head contains $\top$ or body contains $\bot$, and
- removing every $\bot$ in the head, and every $\top$ in the body, of each remaining rule.

It is clear that these steps have no effect on whether a consistent set Y of literals is closed under the program. $\square$

7.2. Proof of Proposition 2

Lemma 3. Let Π be a set of basic constraints, and let X be a consistent set of literals. If X is closed under Π then every subset of X is closed under Π.

Proof. It is easy to see that, for any consistent sets X, Y of literals and any basic formula G, if $Y \subseteq X$ and $Y \models G$ then $X \models G$. $\square$

Proposition 2. Let Π_1, Π_2 be programs such that every rule in Π_2 is a constraint. A consistent set of literals is an answer set for $\Pi_1 \cup \Pi_2$ iff it is an answer set for Π_1 and does not violate Π_2.

Proof. Let X be a consistent set of literals. We need to show that X is an answer for $\Pi_1^X \cup \Pi_2^X$ iff it is an answer set for Π_1^X and does not violate Π_2.

Left-to-right: Assume that X is an answer for $\Pi_1^X \cup \Pi_2^X$. Then X is closed under both Π_1^X and Π_2^X. The second condition means that X does not violate Π_2. It remains to check the minimality of X. Let Y be a subset of X closed under Π_1^X. Since X is closed under Π_2^X, it follows by Lemma 3 that Y is closed under Π_2^X also. Since X is minimal under the sets closed under $\Pi_1^X \cup \Pi_2^X$, it follows that $Y = X$.

Right-to-left: Assume that X is an answer set for Π_1^X and does not violate Π_2. The second condition means that X is closed under Π_2^X. Consequently, X is closed under both Π_1^X and Π_2^X. It remains to check the minimality of X. Let Y

be a subset of X closed under $\Pi_1^X \cup \Pi_2^X$ Then, in particular, Y is closed under Π_1^X. Since X is minimal among such sets, it follows that $Y = X$. $\square$

7.3. Proof of Proposition 3

Lemma 4. Let F, G, H be formulas such that $F \Leftrightarrow G$. If a formula H' is obtained from H by replacing some regular occurrences of F by G then $H \Leftrightarrow H'$.

Proof. By structural induction on H.

Case 1: H is elementary. Then the only regular occurrence of a formula in H is H itself. Consequently either $H = F$ and $H' = G$ or $H' = H$. In both cases, $H \Leftrightarrow H'$.

Case 2: $H = H_1, H_2$. If $H = F$ and $H' = G$ then the assertion is trivial. Otherwise, $H' = H_1', H_2'$ and, by the induction hypothesis, $H_1 \Leftrightarrow H_1'$, $H_2 \Leftrightarrow H_2'$. Then

$$\begin{aligned}
X \models H^Y \;\; &\text{iff} \;\; X \models (H_1, H_2)^Y \\
&\text{iff} \;\; X \models (H_1)^Y, (H_2)^Y \\
&\text{iff} \;\; X \models (H_1)^Y \text{ and } X \models (H_2)^Y \\
&\text{iff} \;\; X \models (H_1')^Y \text{ and } X \models (H_2')^Y \\
&\text{iff} \;\; X \models (H_1')^Y, (H_2')^Y \\
&\text{iff} \;\; X \models (H_1', H_2')^Y \\
&\text{iff} \;\; X \models (H')^Y.
\end{aligned}$$

Case 3: $H = H_1; H_2$. Similar to Case 2.

Case 4: $H = \mathit{not}\, H_1$. If $H = F$ and $H' = G$ then the assertion is trivial. Otherwise, $H' = \mathit{not}\, H_1'$ and, by the induction hypothesis, $H_1 \Leftrightarrow H_1'$. Then

$$\begin{aligned}
X \models H^Y \;\; &\text{iff} \;\; X \models (\mathit{not}\, H_1)^Y \\
&\text{iff} \;\; (\mathit{not}\, H_1)^Y = \top \\
&\text{iff} \;\; Y \not\models (H_1)^Y \\
&\text{iff} \;\; Y \not\models (H_1')^Y \\
&\text{iff} \;\; (\mathit{not}\, H_1')^Y = \top \\
&\text{iff} \;\; X \models (\mathit{not}\, H_1')^Y \\
&\text{iff} \;\; X \models (H')^Y.
\end{aligned}$$

$\square$

Proposition 3. Let Π be a program, and let F, G be a pair of equivalent formulas. Any program obtained from Π by replacing some regular occurrences of F by G is equivalent to Π.

Proof. Assume that Π' can be obtained from Π by replacing some regular occurrences of F by an equivalent formula G, and assume that X is closed under $(\Pi')^Y$. Take any rule $H_1 \leftarrow H_2$ in Π. For the corresponding rule $H_1' \leftarrow H_2'$ in Π', if $X \models (H_2')^Y$ then $X \models (H_1')^Y$. By Lemma 4, H_1 is equivalent to H_1', and H_2 is equivalent to H_2'. Consequently, if $X \models (H_2)^Y$ then $X \models (H_1)^Y$. It follows that X is closed under Π^Y. The proof in the other direction is similar. $\square$

7.4. Proof of Proposition 4

Proposition 4. For any formulas F, G, H,

(i) $F, G \Leftrightarrow G, F$ and $F; G \Leftrightarrow G; F$.

(ii) $(F, G), H \Leftrightarrow F, (G, H)$ and $(F; G); H \Leftrightarrow F; (G; H)$.

(iii) $F, (G; H) \Leftrightarrow (F, G); (F, H)$ and $F; (G, H) \Leftrightarrow (F; G), (F; H)$.

(iv) $not\,(F, G) \Leftrightarrow not\, F; not\, G$ and $not\,(F; G) \Leftrightarrow not\, F, not\, G$.

(v) $not\; not\; not\, F \Leftrightarrow not\, F$.

(vi) $F, \top \Leftrightarrow F$ and $F; \top \Leftrightarrow \top$.

(vii) $F, \bot \Leftrightarrow \bot$ and $F; \bot \Leftrightarrow F$.

(viii) if p is an atom then $p, \neg p \Leftrightarrow \bot$ and $not\, p; not\, \neg p \Leftrightarrow \top$.

(ix) $not\, \top \Leftrightarrow \bot$ and $not\, \bot \Leftrightarrow \top$.

Proof. All parts other than (iv) and (v) follow directly from the definitions. For the first assertion of part (iv),

$$
\begin{aligned}
X \models (not\,(F, G))^Y \;\; &\text{iff } (not\,(F, G))^Y = \top \\
&\text{iff } Y \not\models (F, G)^Y \\
&\text{iff } Y \not\models F^Y, G^Y \\
&\text{iff } Y \not\models F^Y \text{ or } Y \not\models G^Y \\
&\text{iff } (not\, F)^Y = \top \text{ or } (not\, G)^Y = \top \\
&\text{iff } X \models (not\, F)^Y \text{ or } X \models (not\, G)^Y \\
&\text{iff } X \models (not\, F)^Y; (not\, G)^Y \\
&\text{iff } X \models (not\, F; not\, G)^Y.
\end{aligned}
$$

The proof of the second assertion is similar.

For part (v),

$$\begin{aligned} X \models (not\ not\ not\ F)^Y \ &\text{iff}\ (not\ not\ not\ F)^Y = \top \\ &\text{iff}\ Y \not\models (not\ not\ F)^Y \\ &\text{iff}\ (not\ not\ F)^Y = \bot \\ &\text{iff}\ Y \models (not\ F)^Y \\ &\text{iff}\ (not\ F)^Y = \top \\ &\text{iff}\ X \models (not\ F)^Y. \end{aligned}$$

□

7.5. Proof of Proposition 5

Lemma 5. (i) If F is equivalent to a simple conjunction then *not* F is equivalent to a simple disjunction. (ii) If F is equivalent to a simple disjunction then *not* F is equivalent to a simple conjunction.

Proof. Immediate from Proposition 4(i,ii,iv,v) and Lemma 4. □

Proposition 5. Any formula is equivalent to

(i) a formula of the form $F_1; \dots; F_n$ where $n \geq 1$ and each F_i is a simple conjunction, and

(ii) a formula of the form $F_1, \dots, F_n$ where $n \geq 1$ and each F_i is a simple disjunction.

Proof. Both parts are proved simultaneously by structural induction. Every elementary formula is a literal or $\top$ or $\bot$; in the latter cases, use Proposition 4(viii). Otherwise, assume that formulas F and G are each equivalent to formulas of the forms described in parts (i) and (ii) of Proposition 5. We need to check the same for formulas F, G and $F; G$. This follows from Proposition 4(ii,iii) and Lemma 4. We also need to check that *not* F is equivalent to formulas of the forms described in parts (i) and (ii) of Proposition 5. This follows from Proposition 4(iv) and Lemmas 4 and 5. □

7.6. Proof of Proposition 6

Proposition 6.

(i) $F, G \leftarrow H$ is equivalent to

$$F \leftarrow H\,,$$
$$G \leftarrow H\,.$$

(ii) $F \leftarrow G; H$ is equivalent to

$$F \leftarrow G\,,$$
$$F \leftarrow H\,.$$

(iii) $F \leftarrow G, \mathit{not\ not}\, H$ is equivalent to $F; \mathit{not}\, H \leftarrow G\,.$

(iv) $F; \mathit{not\ not}\, G \leftarrow H$ is equivalent to $F \leftarrow \mathit{not}\, G, H\,.$

Proof. Let X, Y be consistent sets of literals.

Part (i): we need to check that X is closed under $F^Y, G^Y \leftarrow H^Y$ iff X is closed under $F^Y \leftarrow H^Y$ and $G^Y \leftarrow H^Y$. This is immediate from the definitions of closure and satisfaction.

The proof of part (ii) is similar.

Part (iii): we need to check that X is closed under

$$F^Y \leftarrow G^Y, (\mathit{not\ not}\, H)^Y \tag{13}$$

iff X is closed under

$$F^Y; (\mathit{not}\, H)^Y \leftarrow G^Y. \tag{14}$$

If $(\mathit{not}\, H)^Y = \top$ then $(\mathit{not\ not}\, H)^Y = \bot$, so that (13), (14) turn into

$$F^Y \leftarrow G^Y, \bot$$

and

$$F^Y; \top \leftarrow G^Y.$$

Clearly X is closed under both rules. Otherwise $(\mathit{not}\, H)^Y = \bot$ and so $(\mathit{not\ not}\, H)^Y = \top$. Then (13), (14) turn into

$$F^Y \leftarrow G^Y, \top$$

and

$$F^Y; \bot \leftarrow G^Y.$$

Clearly X is closed under the first of these rules iff X is closed under the second.

Part (iv): we need to check that X is closed under

$$F^Y; (not\ not\ G)^Y \leftarrow H^Y \tag{15}$$

iff X is closed under

$$F^Y \leftarrow (not\ G)^Y, H^Y. \tag{16}$$

If $G^Y = \bot$, then (15) turns into

$$F^Y; \bot \leftarrow H^Y,$$

while (16) turns into

$$F^Y \leftarrow \top, H^Y.$$

Clearly X is closed under the latter iff it is closed under the former. On the other hand, if $G^Y = \top$, then (15) turns into

$$F^Y; \top \leftarrow H^Y,$$

while (16) turns into

$$F^Y \leftarrow \bot, H^Y.$$

Clearly X is closed under both rules. □

7.7. Proof of Proposition 7

Proposition 7. Any program is equivalent to a set of rules of form (5).

Proof. Propositions 3, 5 and 6(i,ii) show that any program is equivalent to a set of rules of the following form: the head is a simple disjunction, and the body is a simple conjunction. Thus we arrive at a set of rules of the form

$$F_1; \ldots; F_m \leftarrow F_{m+1}, \ldots, F_n \tag{17}$$

where $0 < m < n$ and each F_i has one of the forms

$$L, not\ L, not\ not\ L$$

where L is a literal. Proposition 6(iii,iv) shows that any such rule is equivalent to a rule of the same form (17) in which every F_i has one of the forms

$$L, not\ L\,.$$

□

7.8. Proof of Proposition 8

Lemma 6. For any propositional formula F, interpretation I and consistent set X of literals, $X \models (\tau F)^I$ iff I satisfies F.

Proof. By structural induction on F. If F is atomic then

$$\begin{aligned} X \models (\tau F)^I \ &\text{iff}\ X \models (\mathit{not\ not}\ F)^I \\ &\text{iff}\ (\mathit{not\ not}\ F)^I = \top \\ &\text{iff}\ I \not\models (\mathit{not}\ F)^I \\ &\text{iff}\ (\mathit{not}\ F)^I = \bot \\ &\text{iff}\ I \models F^I \\ &\text{iff}\ I \models F \\ &\text{iff}\ F \in I \\ &\text{iff}\ I \text{ satisfies } F. \end{aligned}$$

If F is $G \wedge H$ then

$$\begin{aligned} X \models (\tau F)^I \ &\text{iff}\ X \models (\tau G, \tau H)^I \\ &\text{iff}\ X \models (\tau G)^I, (\tau H)^I \\ &\text{iff}\ X \models (\tau G)^I \text{ and } X \models (\tau H)^I \\ &\text{iff}\ I \text{ satisfies } G \text{ and } I \text{ satisfies } H \\ &\text{iff}\ I \text{ satisfies } G \wedge H. \end{aligned}$$

In the case when F is $G \vee H$ the proof is similar. If F is $\neg G$ then

$$\begin{aligned} X \models (\tau F)^I \ &\text{iff}\ X \models (\mathit{not}\ \tau G)^I \\ &\text{iff}\ (\mathit{not}\ \tau G)^I = \top \\ &\text{iff}\ I \not\models (\tau G)^I \\ &\text{iff}\ I \text{ does not satisfy } G \\ &\text{iff}\ I \text{ satisfies } \neg G. \end{aligned}$$

□

Proposition 8. For any Lloyd-Topor program Π and any interpretation I, I is a supported model of Π iff I is an answer set for $\tau\Pi$.

Proof. The rules of $(\tau\Pi)^I$ have the form $p \leftarrow (\tau F)^I$, where $p \leftarrow F$ is a rule from Π. Consequently, a consistent set X of literals is closed under $(\tau\Pi)^I$ iff X includes every atom p such that there is a rule $p \leftarrow F$ in Π for which $X \models (\tau F)^I$. According to Lemma 6, the last formula simply says that I satisfies F. It follows that $(\tau\Pi)^I$ has a unique answer set—the set of all atoms p such that there is a rule $p \leftarrow F$ in Π for which I satisfies F. Consequently I is an answer set for $\tau\Pi$ iff it satisfies the following condition: for any atom p, $p \in I$ iff there is a rule $p \leftarrow F$ in Π for which I satisfies F. This is equivalent to saying that I is a supported model of Π. □

7.9. Proof of Proposition 9

For any nonempty set S of interpretations, let $\mathrm{L}(S)$ be the (consistent) set of literals satisfied by every member of S.

The first lemma addresses the four equations in the recursive definition of the translation μ that apply to formulas that begin with *not*.

Lemma 7. For any formula F and nonempty sets S, S' of interpretations such that $S \subseteq S'$, $(\mathit{not}\, F)^{\mathrm{L}(S)} = \top$ iff $\mu(\mathit{not}\, F)$ is true in (I, S', S).

Proof. By structural induction.

Case 1: F is an elementary formula. Then

$$\begin{aligned}
(\mathit{not}\, F)^{\mathrm{L}(S)} = \top \ &\text{iff } \mathrm{L}(S) \not\models F^{\mathrm{L}(S)} \\
&\text{iff } \mathrm{L}(S) \not\models F \\
&\text{iff some member of } S \text{ does not satisfy } F \\
&\text{iff for some } J \in S,\ F \text{ is not true in } (J, S', S) \\
&\text{iff } \mathit{not}\, F \text{ is true in } (I, S', S) \\
&\text{iff } \mu(\mathit{not}\, F) \text{ is true in } (I, S', S).
\end{aligned}$$

Case 2: F is (G, H). By the induction hypothesis, $(\mathit{not}\, G)^{\mathrm{L}(S)} = \top$ iff $\mu(\mathit{not}\, G)$ is true in (I, S', S), and $(\mathit{not}\, H)^{\mathrm{L}(S)} = \top$ iff $\mu(\mathit{not}\, H)$ is true in (I, S', S). Thus,

$$\begin{aligned}
(\mathit{not}\,(G, H))^{\mathrm{L}(S)} = \top \ &\text{iff } \mathrm{L}(S) \not\models (G, H)^{\mathrm{L}(S)} \\
&\text{iff } \mathrm{L}(S) \not\models (G^{\mathrm{L}(S)}, H^{\mathrm{L}(S)}) \\
&\text{iff } \mathrm{L}(S) \not\models G^{\mathrm{L}(S)} \text{ or } \mathrm{L}(S) \not\models H^{\mathrm{L}(S)} \\
&\text{iff } (\mathit{not}\, G)^{\mathrm{L}(S)} = \top \text{ or } (\mathit{not}\, H)^{\mathrm{L}(S)} = \top \\
&\text{iff } \mu(\mathit{not}\, G) \text{ is true in } (I, S', S) \\
&\quad\ \text{or } \mu(\mathit{not}\, H) \text{ is true in } (I, S', S) \\
&\text{iff } \mu(\mathit{not}\, G) \vee \mu(\mathit{not}\, H) \text{ is true in } (I, S', S) \\
&\text{iff } \mu(\mathit{not}\,(G, H)) \text{ is true in } (I, S', S)\,.
\end{aligned}$$

Case 3: F is $(G; H)$. Similar to Case 2.

Case 4: F is $not\,G$. By the induction hypothesis, $(not\,G)^{\mathrm{L}(S)} = \top$ iff $\mu(not\,G)$ is true in (I, S', S). Thus,

$$\begin{aligned}
(not\;not\,G)^{\mathrm{L}(S)} = \top \;\; & \text{iff } \mathrm{L}(S) \not\models (not\,G)^{\mathrm{L}(S)} \\
& \text{iff } (not\,G)^{\mathrm{L}(S)} \neq \top \\
& \text{iff } \mu(not\,G) \text{ is not true in } (I, S', S) \\
& \text{iff } \neg\mu(not\,G) \text{ is true in } (I, S', S) \\
& \text{iff } \mu(not\;not\,G) \text{ is true in } (I, S', S)\,.
\end{aligned}$$

□

Lemma 8. For any formula F and nonempty sets S, S' of interpretations such that $S \subseteq S'$, $\mathrm{L}(S') \models F^{\mathrm{L}(S)}$ iff μF is true in (I, S', S).

Proof. By structural induction.

Case 1: F is elementary. Then

$$\begin{aligned}
\mathrm{L}(S') \models F^{\mathrm{L}(S)} \;\; & \text{iff } \mathrm{L}(S') \models F \\
& \text{iff every member of } S' \text{ satisfies } F \\
& \text{iff for every } J \in S',\ F \text{ is true in } (J, S', S) \\
& \text{iff } \mathsf{B}F \text{ is true in } (I, S', S) \\
& \text{iff } \mu F \text{ is true in } (I, S', S)\,.
\end{aligned}$$

Case 2: F is (G, H). By the induction hypothesis, $L(S') \models G^{\mathrm{L}(S)}$ iff μG is true in (I, S', S), and $L(S') \models H^{\mathrm{L}(S)}$ iff μH is true in (I, S', S). Thus,

$$\begin{aligned}
\mathrm{L}(S') \models (G, H)^{\mathrm{L}(S)} \;\; & \text{iff } \mathrm{L}(S') \models (G^{\mathrm{L}(S)}, H^{\mathrm{L}(S)}) \\
& \text{iff } \mathrm{L}(S') \models G^{\mathrm{L}(S)} \text{ and } \mathrm{L}(S') \models H^{\mathrm{L}(S)} \\
& \text{iff both } \mu G \text{ and } \mu H \text{ are true in } (I, S', S) \\
& \text{iff } \mu G \wedge \mu H \text{ is true in } (I, S', S) \\
& \text{iff } \mu(G, H) \text{ is true in } (I, S', S)\,.
\end{aligned}$$

Case 3: F is $(G; H)$. Similar to Case 2.

Case 4: F is $not\,G$. Since $\mathrm{L}(S') \models (not\,G)^{\mathrm{L}(S)}$ iff $(not\,G)^{\mathrm{L}(S)} = \top$, the claim follows from the previous lemma. □

Lemma 9. For any program Π and nonempty sets S, S' of interpretations such that $S \subseteq S'$, $\mathrm{L}(S')$ is closed under $\Pi^{\mathrm{L}(S)}$ iff every formula in $\mu\Pi$ is true in (I, S', S).

Proof. By definition, $\mathrm{L}(S')$ is closed under $\Pi^{\mathrm{L}(S)}$ iff, for every rule $F \leftarrow G \in \Pi$, $\mathrm{L}(S') \models F^{\mathrm{L}(S)}$ if $\mathrm{L}(S') \models G^{\mathrm{L}(S)}$. The previous lemma shows that $\mathrm{L}(S') \models F^{\mathrm{L}(S)}$

iff μF is true in (I, S', S), and that $\mathrm{L}(S') \models G^{\mathrm{L}(S)}$ iff μG is true in (I, S', S). Hence,

$$\begin{aligned}
\mathrm{L}(S') \models F^{\mathrm{L}(S)} \text{ if } \mathrm{L}(S') \models G^{\mathrm{L}(S)} \ & \text{iff } \mu F \text{ is true in } (I, S', S) \text{ if } \mu G \text{ is} \\
& \text{iff } \mu G \supset \mu F \text{ is true in } (I, S', S) \\
& \text{iff } \mu(F \leftarrow G) \text{ is true in } (I, S', S)\,.
\end{aligned}$$

Since $\mu\Pi = \{\mu(F \leftarrow G) : F \leftarrow G \in \Pi\}$, we're done. □

Proposition 9. For any program Π, consistent set X of literals, and interpretation I, X is an answer set for Π iff $(I, \mathrm{Mod}(X))$ is a model of $\mu\Pi$. Moreover, every consistent model of $\mu\Pi$ can be represented in the form $(I, \mathrm{Mod}(X))$, where I is an interpretation and X is a consistent set of literals.

Proof. We show the second part first. Assume that (I, S) is a consistent model of $\mu\Pi$. It follows that every formula in $\mu\Pi$ is true in (I, S, S). So by Lemma 9, $\mathrm{L}(S)$ is closed under $\Pi^{\mathrm{L}(S)}$. Let $X = \mathrm{L}(S)$. Notice that $X = \mathrm{L}(\mathrm{Mod}(X))$, so we know by Lemma 9 that every formula in $\mu\Pi$ is true in $(I, \mathrm{Mod}(X), S)$. Next notice that $S \subseteq \mathrm{Mod}(L(S))$. That is, $S \subseteq \mathrm{Mod}(X)$. Since (I, S) is a model of $\mu\Pi$, it follows that $S = \mathrm{Mod}(X)$.

For the right-to-left direction of the first part, assume that $(I, \mathrm{Mod}(X))$ is a model of $\mu\Pi$, where X is consistent set of literals. It follows that every formula in $\mu\Pi$ is true in $(I, \mathrm{Mod}(X), \mathrm{Mod}(X))$, and since $X = \mathrm{L}(\mathrm{Mod}(X))$, we know by Lemma 9 that X is closed under Π^X. Let Y be a subset of X that is closed under Π^X. Since $Y = \mathrm{L}(\mathrm{Mod}(Y))$, we know by Lemma 9 that every formula in $\mu\Pi$ is true in $(I, \mathrm{Mod}(Y), \mathrm{Mod}(X))$. Since $\mathrm{Mod}(X) \subseteq \mathrm{Mod}(Y)$ and $(I, \mathrm{Mod}(X))$ is a model of $\mu\Pi$, we can conclude that $X = Y$, which shows that X is an answer set for Π.

Now assume that X is an answer set for Π. It follows that X is closed under Π^X. By Lemma 9, every formula in $\mu\Pi$ is true in $(I, \mathrm{Mod}(X), \mathrm{Mod}(X))$. Let S be a superset of $\mathrm{Mod}(X)$ such that every formula in $\mu\Pi$ is true in $(I, S, \mathrm{Mod}(X))$. By Lemma 9, $\mathrm{L}(S)$ is closed under Π^X. Since $\mathrm{L}(S) \subseteq X$ and X is an answer set for Π, we conclude that $\mathrm{L}(S) = X$. It follows that $S \subseteq \mathrm{Mod}(X)$, and since $\mathrm{Mod}(X) \subseteq S$, $\mathrm{Mod}(X) = S$. We can conclude that $(I, \mathrm{Mod}(X))$ is a model of $\mu\Pi$. □

8. Conclusions

We showed how the answer set semantics can be extended to a class of logic programs with nested expressions in the bodies and heads of rules. In the special case of disjunctive logic programs, the answer sets according to the new definition are exactly the consistent answer sets according to the traditional definition. In published papers, inconsistent answer sets were treated in different ways; the new definition does not allow for an inconsistent answer set at all. Our proposal can be used as a specification for extending systems like SMODELS and DLV to programs with nested expressions.

The study of equivalent transformations of programs with nested expressions shows that such expressions can be always eliminated, altough this process may lead to the emergence of additional occurrences of negation as failure in the heads of rules.

The proposed definition is related to the completion semantics for programs with formulas in the bodies of rules due to Lloyd and Topor and to the logic of minimal belief and negation as failure.

Acknowledgements

We are grateful to Esra Erdem, Michael Gelfond, Katsumi Inoue, David Pearce, David S. Warren and Phoebe Weidmann, and to anonymous referees, for useful comments on the subject of this paper. This work was partially supported by National Science Foundation under grants IRI-9306751 and IRI-9732744.

References

[1] J.J. Alferes, J.A. Leite, L.M. Pereira, H. Przymusinska, and T.C. Przymusinski. Dynamic logic programming. In Anthony Cohn, Lenhart Schubert, and Stuart Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proc. Sixth Int'l Conf.*, pages 98–109, 1998.

[2] 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.

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

[4] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding planning problems in non-monotonic logic programs. In *Proc. European Conference on Planning 1997*, pages 169–181, 1997.

[5] Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The KR system DLV: Progress report, comparisons and benchmarks. In Anthony Cohn, Lenhart Schubert, and Stuart Shapiro, editors, *Proc. Sixth Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 406–417, 1998.

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

[7] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[8] Heinrich Herre and Gerd Wagner. Stable models are generated by a stable chain. *Journal of Logic Programming*, 30:165–177, 1997.

[9] C.A.R. Hoare. Programs are predicates. In *Proc. Int'l Conf. on Fifth Generation Computer Systems*, pages 211–218, 1992.

[10] Katsumi Inoue and Chiaki Sakama. On positive occurrences of negation as failure. In *Proc. Fourth Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 293–304, 1994.

[11] Katsumi Inoue and Chiaki Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35:39–78, 1998.

[12] Vladimir Lifschitz. Minimal belief and negation as failure. *Artificial Intelligence*, 70:53–72, 1994.

[13] Vladimir Lifschitz. Foundations of logic programming. In Gerhard Brewka, editor, *Principles of Knowledge Representation*, pages 69–128. CSLI Publications, 1996.

[14] Vladimir Lifschitz. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*. Springer Verlag, 1999. To appear.

[15] Vladimir Lifschitz and Thomas Woo. Answer sets in general nonmonotonic reasoning (preliminary report). In Bernhard Nebel, Charles Rich, and William Swartout, editors, *Proc. Third Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 603–614, 1992.

[16] Fangzhen Lin and Yoav Shoham. A logic of knowledge and justified assumptions. *Artificial Intelligence*, 57:271–289, 1992.

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

[18] Victor Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*. Springer Verlag, 1999. To appear.

[19] 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.

[20] David Pearce. A new logical characterization of stable models and answer sets. In Jürgen Dix, Luis Pereira, and Teodor Przymusinski, editors, *Non-Monotonic Extensions of Logic*

Programming (Lecture Notes in Artificial Intelligence 1216), pages 57–70. Springer-Verlag, 1997.

[21] V.S. Subrahmanian and Carlo Zaniolo. Relating stable models and AI planning domains. In *Proc. ICLP-95*, 1995.