A Model-Theoretic Counterpart of Loop Formulas

Joohyung Lee
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712, USA
appsmurf@cs.utexas.edu

Abstract

In an important recent paper, Lin and Zhao introduced the concept of a loop formula, and showed that the answer sets for a logic program are exactly the models of Clark's completion of the program that satisfy the loop formulas. Just as supported sets are a model-theoretic account of completion, "externally supported" sets, defined in this paper, are a model-theoretic counterpart of loop formulas. This reformulation of loop formulas shows that they are related to assumption sets (Saccá and Zaniolo) and to unfounded sets (Van Gelder, Ross and Schlipf; Leone, Rullo and Scarcello), invented many years earlier. Other contributions of this paper include a simplification of the definition of a loop, extending it to programs with classical negation and infinite programs, and a generalization of the definition of a loop formula.

1 Introduction

The completion semantics [Clark, 1978] and the answer set semantics (also known as the stable model semantics) [Gelfond and Lifschitz, 1988] are two well-known proposals for defining the meaning of negation as failure in logic programs.[1] The former is attractive because it is simply a translation from logic programs to classical logic, but it sometimes gives unintuitive results [Przymusinski, 1989, Section 4.1]. It is well known that an answer set for a logic program is also a model of its completion, while the converse, generally, does not hold.

By adding so-called "loop formulas" to completion, Lin and Zhao [2004] defined a set of formulas whose models are exactly the answer sets. The idea comes from observing that "loops" are related to the mismatch between the models of the completion and the answer sets. Intuitively, every atom in an answer set can be "derived" from a program in a finite number of steps. The atoms in a loop that are not "supported from outside" cannot be derived in a finite number of steps, but they do not contradict the completion of the program.

The Lin/Zhao theorem allows us to compute answer sets using SAT solvers, which has led to the implementation of SAT-based answer set solvers ASSAT[2] [Lin and Zhao, 2004] and CMODELS[3] [Giunchiglia *et al.*, 2004]. These systems turn a logic program into a propositional theory and then call SAT solvers, such as CHAFF, SATO and RELSAT, to find its models, which in turn correspond to the answer sets. Thanks to recent progress with SAT solvers, in several benchmark examples they found answer sets faster than other answer set solvers such as SMODELS[4] and DLV[5].

A program that has no loops is called "tight," and for tight programs the completion semantics and the answer set semantics are equivalent to each other. This fact was discovered earlier by Fages [1994], and then was generalized and extended by Erdem and Lifschitz [2003] to programs with nested expressions (in the sense of [Lifschitz *et al.*, 1999]) in the bodies of rules.

Lee and Lifschitz [2003] showed that program completion, loop formulas and the Lin/Zhao theorem can be extended to disjunctive logic programs and, more generally, to arbitrary programs with nested expressions. As a consequence, the concept of a tight program and Fages' theorem are extended to disjunctive programs as well.

Yet the previous work on loop formulas has been limited to *finite* programs *without classical negation.* In this paper, we lift these limitations. First, to account for programs that allow classical negation, we propose a model-theoretic counterpart of loop formulas—the concept of an "externally supported" set. It is similar to the model-theoretic counterpart of Clark's completion—the concept of a supported set introduced in [Apt *et al.*, 1988]. Interestingly, this reformulation of loop formulas shows that they are related to assumption sets [Saccá and Zaniolo, 1990] and to unfounded sets [Van Gelder *et al.*, 1991; Leone *et al.*, 1997] invented many years earlier. Second, we extend the concept of a loop formula to programs with nested expressions. Finally, we extend the notion of a loop and the theorem on loop formulas to infinite programs.

In the next section we introduce the concept of an exter-

[1] As the answer set semantics is usually defined for propositional (grounded) logic programs only, in this paper we limit our attention to such programs, and refer to the propositional case of Clark's completion.

[2] http://assat.cs.ust.hk/ .

[3] http://www.cs.utexas.edu/users/tag/cmodels/ .

[4] http://www.tcs.hut.fi/Software/smodels/ .

[5] http://www.dbai.tuwien.ac.at/proj/dlv/ .

nally supported set, extend the notion of a loop to infinite programs, state our main theorem, and discuss its relation to Fages' theorem about tight programs. In Section 3 we use the main theorem to define two transformations turning a logic program into an equivalent propositional theory. In Section 4, we relate externally supported sets to assumption sets and to unfounded sets, and compare our reformulation of loop formulas with the original definition by Lin and Zhao. The main theorem is generalized to arbitrary programs with nested expressions in Section 5.

2 Externally Supported Sets

2.1 Nondisjunctive Case

We begin with a review of the answer set semantics for nondisjunctive programs given in [Gelfond and Lifschitz, 1991, Section 2]. [6] The words *atom* and *literal* are understood here as in propositional logic; we call the negation sign $\neg$ in negative literals *classical negation*, to distinguish it from the symbol for negation as failure (*not*).

A *nondisjunctive rule* is an expression of the form

$$l_1 \leftarrow l_2, \ldots, l_m, \mathit{not}\ l_{m+1}, \ldots, \mathit{not}\ l_n \qquad (1)$$

($1 \leq m \leq n$) where all l_i are literals. We will often write (1) in the form

$$l_1 \leftarrow B, F \qquad (2)$$

where B is $l_2, \ldots, l_m$, and F is $\mathit{not}\ l_{m+1}, \ldots, \mathit{not}\ l_n$, and we will sometimes identify B with the set $\{l_2, \ldots, l_m\}$.

A *nondisjunctive (logic) program* is a set of rules of the form (1).

We say that a set X of literals *satisfies* the body B, F of rule (2) (symbolically $X \models B, F$) if $l_2, \ldots, l_m \in X$, and $l_{m+1}, \ldots, l_n \notin X$. We say that X *satisfies* a nondisjunctive program Π (symbolically $X \models \Pi$) if, for every rule (2) of that program, $l_1 \in X$ whenever X satisfies B, F.

The reduct Π^X of a nondisjunctive program Π with respect to a set X of literals is obtained from Π by deleting each rule (2) such that $X \not\models F$, and replacing each remaining rule (2) by $l_1 \leftarrow B$. A consistent set X of literals is an *answer set* for Π if X is minimal among the sets of literals that satisfy Π^X.

We say that a set Y of literals is *supported* by Π w.r.t. a set X of literals if there is a rule (2) in Π such that $l_1 \in Y$, and $X \models B, F$. Informally, such a rule (2) characterizes a source of support for the literal l_1 in Y.[7]

Now we make the notion of a supported set slightly stronger. We will say that a set Y of literals is *externally supported* by Π w.r.t. X if there is a rule (2) in Π such that $l_1 \in Y$, $X \models B, F$, and $B \cap Y = \emptyset$. Informally, the new, third condition ensures that the support for the literal l_1 in Y comes from *outside* of Y.

[6] Here we do not allow inconsistent answer sets.

[7] One may notice that this definition is slightly different from the usual definition of a supported set, which says that a set X of literals is supported by Π if, for every literal $l_1 \in X$, there is a rule (2) in Π such that $X \models B, F$ [Apt *et al.*, 1988]. Observe that X is supported by Π under the usual definition iff every singleton subset of X is supported by Π w.r.t. X under the new definition.

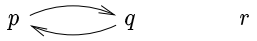

Figure 1: The dependency graph of Π_2

For example, consider the following program Π_1:

$$p \leftarrow q \qquad q \leftarrow p$$

Each of the sets $\{p\}$ and $\{q\}$ is externally supported (hence also supported) by Π_1 w.r.t. $\{p, q\}$, while $\{p, q\}$ is supported but not externally supported by Π_1 w.r.t. $\{p, q\}$. On the other hand, let Π_2 be the following program that adds two rules to Π_1:

$$p \leftarrow q \qquad q \leftarrow p$$
$$p \leftarrow \mathit{not}\ r \qquad r \leftarrow \mathit{not}\ p.$$

Then every nonempty subset of $\{p, q\}$ is externally supported by Π_2 w.r.t. $\{p, q\}$.

The *positive dependency graph* of Π is the directed graph such that

- its vertices are the literals occurring in Π, and
- its edges go from l_1 to $l_2, \ldots l_m$ for each rule (1) of Π.

When Π is finite, a nonempty set L of literals is called a *loop* of Π if, for every pair l_1, l_2 of literals in L, there exists a path from l_1 to l_2 in the positive dependency graph of Π such that all vertices in this path belong to L.[8] In other words, a nonempty set L of literals is a loop of Π iff the subgraph of the positive dependency graph of Π induced by L is strongly connected. Note that, for every literal l that occurs in Π, the singleton set $\{l\}$ is a loop, according to this definition.[9]

The positive dependency graph of Π_2, shown in Figure 1, has four loops: $\{p\}$,$\{q\}$,$\{r\}$,$\{p, q\}$.

Main Theorem for Finite Nondisjunctive Programs For any finite nondisjunctive program Π and any consistent set X of literals, the following conditions are equivalent:

(a) X is an answer set for Π.

(b) X satisfies Π, and every set of literals that has a common element with X is externally supported by Π w.r.t. X.

(c) X satisfies Π, and every loop of Π that is contained in X is externally supported by Π w.r.t. X.

Out of the three implications (a) to (b), (b) to (c), (c) to (a), the second is obvious, because every loop that is contained in X has a common element with X. Program Π_2 above has two answer sets: $\{p, q\}$ and $\{r\}$. Since (a) implies (b), it follows that every set of atoms that has a common element with $\{p, q\}$ is externally supported w.r.t. $\{p, q\}$, and every set of atoms that has a common element with $\{r\}$ is externally supported w.r.t. $\{r\}$. On the other hand, since (c) implies (a), to show that $\{p, q\}$ is an answer set we only need to check that every loop which is a subset of $\{p, q\}$ (that is, each of $\{p\}$, $\{q\}$, $\{p, q\}$) is externally supported w.r.t. $\{p, q\}$. Similarly, to show that $\{r\}$ is an answer set, we only need to check that $\{r\}$ is externally supported w.r.t. $\{r\}$.

[8] Note that we do allow paths of length 0.

[9] Thus our definition is slightly different from the definition by Lin and Zhao (See Section 4 for detail). The example of a singleton loop shows that a loop of Π does not necessarily correspond to a loop (or cycle) of the positive dependency graph of Π in the sense of graph theory.

The equivalence between (a) and (b) is a generalization of a theorem from [Saccá and Zaniolo, 1990] as we will discuss in Section 4.1, and the equivalence between (a) and (c) is a model-theoretic account of loop formulas as we will discuss in Section 3.

2.2 Extension to Programs in Canonical Form

We will extend the main theorem to finite programs with nested expressions [Lifschitz *et al.*, 1999]. In this section, for simplicity, instead of arbitrary rules with nested expressions, we consider rules of the form

$$l_1; \dots; l_k \leftarrow l_{k+1}, \dots, l_m, \mathit{not}\ l_{m+1}, \dots, \mathit{not}\ l_n, \quad \mathit{not\ not}\ l_{n+1}, \dots, \mathit{not\ not}\ l_p \tag{3}$$

($0 \leq k \leq m \leq n \leq p$) where all l_i are literals.[10] We will call such rules *canonical*, and will often write (3) in the form

$$A \leftarrow B, F \tag{4}$$

where A is $l_1, \dots, l_k$, B is $l_{k+1}, \dots, l_m$, and F is

$$\mathit{not}\ l_{m+1}, \dots, \mathit{not}\ l_n, \mathit{not\ not}\ l_{n+1}, \dots, \mathit{not\ not}\ l_p.$$

We will sometimes identify A with the set $\{l_1, \dots, l_k\}$ and B with the set $\{l_{k+1}, \dots, l_m\}$.

A *canonical program* is a set of rules of the form (3).

The definition of satisfaction given in Section 2.1 is extended to canonical programs as follows. We say that a set X of literals *satisfies* the body B, F of rule (4) if $l_{k+1}, \dots, l_m \in X$, $l_{m+1}, \dots, l_n \notin X$, and $l_{n+1}, \dots, l_p \in X$. X *satisfies* a program if, for every rule (4) of that program, at least one of the literals in A belongs to X whenever X satisfies B, F.

The reduct Π^X of a canonical program Π with respect to a set X of literals is obtained from Π by deleting each rule (4) such that $X \not\models F$, and replacing each remaining rule (4) by $A \leftarrow B$. A consistent set X of literals is an *answer set* for Π if X is minimal among the sets of literals that satisfy Π^X.

Given a canonical program Π and a set X of literals, we will say that a set Y of literals is *externally supported* by Π w.r.t. X if there is a rule (4) in Π such that $A \cap Y \neq \emptyset$, $X \models B, F$, $B \cap Y = \emptyset$, and $X \cap (A \setminus Y) = \emptyset$. The last condition is suggested by the extension of the definition of a supported set to disjunctive programs proposed in [Baral and Gelfond, 1994] and [Inoue and Sakama, 1998].

The definition of a positive dependency graph is extended to the general case straightforwardly: for each rule (3) in Π, the edges of the graph go from each literal l_i ($0 < i \leq k$) to each literal l_j ($k < j \leq m$). The definition of a loop from Section 2.1 remains the same for arbitrary finite programs.

The theorem from Section 2.1 remains true if we replace "nondisjunctive" in its statement with "canonical."

For example, let Π_3 be the program

$$p\ ;\ s \leftarrow q \qquad q \leftarrow p \qquad p\ ;\ r \leftarrow \mathit{not}\ s,$$

which has two answer sets: $\{p, q\}$ and $\{r\}$. The comments about these answer sets at the end of Section 2.2 apply to program Π_3 as well.

[10] Any program with nested expressions can be turned into an equivalent program whose rules have the form (3), or equivalently $l_1; \dots; l_k; \mathit{not}\ l_{n+1}; \dots; \mathit{not}\ l_p \leftarrow l_{k+1}, \dots, l_m, \mathit{not}\ l_{m+1}, \dots, \mathit{not}\ l_n$ [Lifschitz *et al.*, 1999, Proposition 6 (iii)]. In Section 5, the main theorem is extended to arbitrary rules with nested expressions.

2.3 Extension to Infinite Programs

So far we restricted our attention to finite programs only. Indeed, unless we modify the definition of a loop, the theorem in Section 2.1 does not hold for infinite programs: conditions (a) and (b) are equivalent to each other even for infinite programs, but condition (c) turns out weaker than the others. For example, consider the following infinite program Π_4:

$$p_i \leftarrow p_{i+1} \qquad (i > 0),$$

where each p_i is an atom. The program has no loops (in the sense of Section 2.1) other than singletons. Besides $\emptyset$, which is the only answer set for Π_4, there is one more set that satisfies condition (c): $\{p_1, p_2, \dots\}$.

Looking at the example, we observe that condition (c) is weaker than condition (b) in that it does not account for infinite paths in the positive dependency graph which do not correspond to loops, while condition (b) tells us that the sets that correspond to such infinite paths should also be externally supported. We propose to modify the definition of a loop as follows.

Given a (possibly infinite) program Π and a set L of literals, we say that L is *unbounded* if, for every literal $l \in L$, there exists an infinite path in L in the positive dependency graph of Π that starts from l and does not visit the same vertex more than once. A nonempty set L of literals is called a *loop* of Π if L is unbounded, or for every pair l_1, l_2 of literals in L, there exists a path from l_1 to l_2 in the positive dependency graph of Π such that all vertices in this path belong to L. When Π is finite, no set of literals that occur in Π is unbounded, so that this definition of a loop reduces to the earlier definition.

In program Π_4 above, every set $\{p_i, p_{i+1}, \dots\}$ $(i > 0)$ is a loop according to this definition, since it is unbounded. But it is not externally supported by Π_4 w.r.t. $\{p_1, p_2, \dots\}$.

The theorem from Section 2.1 remains true if we replace "finite nondisjunctive" with "canonical":

2.4 Tight Programs

The notion of a loop helps us simplify the definition of a "tight" program [Erdem and Lifschitz, 2003]. Let Π be any canonical program. We will say that a loop of Π is *trivial* if the loop consists of a single literal such that the positive dependency graph of Π does not contain an edge from the literal to itself. For example, in program Π_4 above, every singleton loop is trivial, while every loop $\{p_i, p_{i+1}, \dots\}$ $(i > 0)$ is not. We say that Π is *absolutely tight* if every loop of Π is trivial. Let Π_X be the set of rules of Π whose heads and bodies are satisfied by X. Program Π is *tight on* a set X of literals if the subgraph of the positive dependency graph of Π_X induced by X has no loops other than trivial ones. These definitions are more general than the corresponding definitions in [Lee and Lifschitz, 2003] in that they are applicable to infinite programs and to programs with classical negation as well. We get the following corollaries to the main theorem, which generalize Propositions 2 and 4 from [Lee and Lifschitz, 2003].

Corollary 1 *For any absolutely tight canonical program* Π *and any consistent set* X *of literals,* X *is an answer set for* Π *iff* X *satisfies* Π *and* X *is supported by* Π.

Corollary 2 *For any canonical program* Π *and any consistent set* X *of literals such that* Π *is tight on* X*,* X *is an answer set for* Π *iff* X *satisfies* Π *and* X *is supported by* Π.

3 Translating Logic Programs into Propositional Logic

In application to finite programs without classical negation, conditions (b) and (c) of the main theorem suggest ways to turn a program into an equivalent propositional theory. This translation is closely related to the Lin/Zhao theorem on loop formulas, as we will see in Section 4.

Consider a finite program Π whose rules have the form (4) where all l_i are atoms. For any set Y of atoms, the *external support formula* for Y is the disjunction of the conjunctions

$$B \wedge F \wedge \bigwedge_{p \in A \setminus Y} \neg p \tag{5}$$

for all rules (4) of Π such that $A \cap Y \neq \emptyset$, and $B \cap Y = \emptyset$. We will denote the external support formula by $ES_{\Pi,Y}$.

Condition (b) of the main theorem suggests the propositional theory $T_b(\Pi)$ which consists of

(i) the implications [11]

$$B \wedge F \supset A \tag{6}$$

for all rules (4) in Π, and

(ii) the implications $\bigvee_{p \in Y} p \supset ES_{\Pi,Y}$ for all sets Y of atoms that occur in Π.

On the other hand, condition (c) of the main theorem suggests the propositional theory $T_c(\Pi)$, which consists of

(i) the implications (6) for all rules (4) in Π, and

(ii) the implications

$$\bigwedge_{p \in L} p \supset ES_{\Pi,L} \tag{7}$$

for all loops L of Π.

Formula (7) is called the *conjunctive loop formula* of L.

Corollary 3 *For any finite program* Π *without classical negation and any set* X *of atoms,* X *is an answer set for* Π *iff* X *is a model of* $T_b(\Pi)$ *iff* X *is a model of* $T_c(\Pi)$.

Note that the number of loops is exponential in the worst case. But we cannot do much about this: Lifschitz and Razborov [2004] showed that any equivalent translation from logic programs to propositional formulas involves a significant increase in size assuming a conjecture from the theory of computational complexity which is widely believed to be true.

4 Unfounded Sets and Disjunctive Loop Formulas

Since conditions (b) and (c) in the statement of the main theorem are equivalent to each other, any intermediate condition between the two also characterizes answer sets. In this section, we study two such conditions related to some earlier work.

[11] We identify '*not*' with '$\neg$', ',' with '$\wedge$', and ';' with '$\vee$'.

4.1 Relation to Unfounded Sets

The first condition is

(d) X satisfies Π, and every nonempty subset of X is externally supported by Π w.r.t. X.

Since (b) implies (d), (d) implies (c), and (a),(b),(c) are equivalent to each other, it is clear that all these conditions are equivalent to each other.

In fact, the equivalence between conditions (a) and (d) was established before for the special case when the rules of the program have the form (3) with $p = n$. It is related to the notion of an unfounded set originally introduced in [Van Gelder *et al.*, 1991] to characterize negative conclusions under the well-founded semantics. Saccá and Zaniolo [1990] showed that answer sets for a nondisjunctive program can be characterized in terms of unfounded sets.[12] Leone *et al.* [1997] extended the notion of an unfounded set and the theorem by Saccá and Zaniolo to disjunctive programs.

Their definition can be further extended to programs with rules of the form (4) as follows. A set Y of literals is *unfounded* by a program Π w.r.t. a set X of literals if, for each rule (4) in Π such that $A \cap Y \neq \emptyset$, at least one of the following conditions holds: $X \not\models B, F$, $B \cap Y \neq \emptyset$, or $X \cap (A \setminus Y) \neq \emptyset$.

It is easy to see that Y is not unfounded w.r.t. X iff Y is externally supported w.r.t. X in the sense of Section 2.2.

A set X of literals is called *unfounded-free* if it does not have any nonempty subset that is unfounded w.r.t. X. The equivalence between conditions (a) and (d) stated above can be reformulated as follows: X is an answer set for Π iff X satisfies Π and X is unfounded-free. This is a generalization of Corollary 2 from [Saccá and Zaniolo, 1990] and Theorem 4.6 from [Leone *et al.*, 1997] to programs with rules of the form (3).

For finite programs Π without classical negation, condition (d) suggests the following translation T_d into propositional logic, which consists of

(i) the implications (6) for all rules (4) in Π, and

(ii) the implications (7) for all nonempty sets L of atoms that occur in Π.

Note that $T_d(\Pi)$ is a superset of $T_c(\Pi)$. In (ii), L is not empty. If it were, then the implication would be unsatisfiable.

Corollary 4 *For any finite program without classical negation, a set of atoms is an answer set for* Π *iff it is a model of* $T_d(\Pi)$.

4.2 Relation to Disjunctive Loop Formulas

Here is another condition intermediate between (b) and (c):

(e) X satisfies Π, and, for every loop $L \subseteq X$, every set of literals that has a common element with L is externally supported by Π w.r.t. X.

Given a finite program Π whose rules have the form (4) where all l_i are atoms, the propositional theory $T_e(\Pi)$ consists of

[12] Their theorem refers to "assumption sets" rather than "unfounded sets." But as the authors noted, the two notions coincide as far as the theorem is concerned.

(i) the implications (6) for all rules (4) in Π, and

(ii) the implications

$$\bigvee_{p\in L} p \ \supset \ ES_{\Pi,L} \tag{8}$$

for all loops L of Π.

Formula (8) is called the *disjunctive loop formula* of L.

Corollary 5 *For any finite program Π without classical negation, a set of atoms is an answer set for Π iff it is a model of $T_e(\Pi)$.*

In fact, if Π is nondisjunctive, (8) is essentially the loop formula defined by Lin and Zhao [2004]. In the rest of this section, we give a more precise description of the relationship between this special case of Corollary 5 and the Lin/Zhao theorem.

Note first that when Π is nondisjunctive, $T_e(\Pi)$ consists of

(i) the implications

$$B \wedge F \ \supset \ l_1 \tag{9}$$

for all rules (2) in Π, and

(ii) the implications

$$\bigvee_{p\in L} p \ \supset \bigvee_{\substack{l_1\leftarrow B,F \ \in \ \Pi \\ l_1\in L,\ B\cap L=\emptyset}} B \wedge F. \tag{10}$$

for all loops L of Π.

We now review the Lin/Zhao theorem. The completion of the same program Π, $Comp(\Pi)$, consists of the equivalences

$$l_1 \equiv \bigvee_{l_1\leftarrow B,F \ \in \ \Pi} B \wedge F$$

for all atoms l_1 that occur in Π.

By $LF(\Pi)$ we denote the set of the disjunctive loop formulas (10) for all non-trivial loops L.

Theorem 1 *[Lin and Zhao, 2004, Theorem 1] For any finite nondisjunctive program Π without classical negation, a set of atoms is an answer set for Π iff it is a model of $Comp(\Pi) \cup LF(\Pi)$.*

To see why $Comp(\Pi) \cup LF(\Pi)$ is equivalent to $T_e(\Pi)$, observe first that $Comp(\Pi)$ can be rewritten as the set of implications "right-to-left"

$$\left(\bigvee_{l_1\leftarrow B,F \ \in \ \Pi} B \wedge F\right) \supset l_1, \tag{11}$$

and "left-to-right"

$$l_1 \supset \bigvee_{l_1\leftarrow B,F \ \in \ \Pi} B \wedge F. \tag{12}$$

Implications (11) can be further broken into implications (9). In implications (12), every singleton set $\{l_1\}$ is a loop. If the loop is non-trivial, then $LF(\Pi)$ contains the corresponding implication (10), which is stronger than (12). Consequently, in the presence of $LF(\Pi)$, implications (12) for all non-trivial loops $\{l_1\}$ can be dropped. On the other hand, if $\{l_1\}$ is trivial, then $l_1 \notin B$, so that (12) coincides with (10) (where $L = \{l_1\}$). To sum up, $Comp(\Pi) \cup LF(\Pi)$ can be equivalently rewritten as the set consisting of formulas (9), formulas (10) for trivial loops, and formulas (10) for non-trivial loops (they form $LF(\Pi)$). This set is exactly $T_e(\Pi)$.

5 Loop Formulas for Programs with Nested Expressions

Lifschitz *et al.*[1999] extended the answer set semantics to programs with nested expressions. "Formulas" defined in that paper allow negation as failure (not), conjunction (,) and disjunction (;) to be nested arbitrarily. A program with nested expressions is a set of rules of which both heads and bodies are formulas. The following program Π_5, for instance, is a program with nested expressions which is not in canonical form because the first rule has a disjunction in its body.

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

For the semantics of such programs, along with the definition of satisfaction, we refer the reader to Section 2 of [Erdoğan and Lifschitz, 2004]. Recall that an occurrence of a formula F in a formula G is *singular* if the symbol before this occurrence is $\neg$; otherwise, the occurrence is *regular* [Lifschitz *et al.*, 1999]. For any formula G, by $poslit(G)$ we denote the set of all literals having a regular occurrence in G that is not in the scope of negation as failure. For instance, $poslit(p, not\ q, (not\ not\ r; \neg s)) = \{p, \neg s\}$.

For a set Y of literals, by $F^Y_\bot$ we denote the formula obtained from a formula F by replacing all regular occurrences of literals from Y that are not in the scope of negation as failure with $\bot$; by $\Pi^Y_\bot$ we denote the program obtained from a program Π by the same substitution. In application to canonical programs, this operation is closely related to the concept of external support:

Proposition 1 *Let Π be a canonical program, and X a set of literals satisfying Π. For any set Y of literals, Y is externally supported by Π w.r.t. X iff X does not satisfy $\Pi^Y_\bot$.*

For example, for program Π_2 in Section 2.1 we checked that $\{p, q\}$ is externally supported w.r.t. $\{p, q\}$. On the other hand, the program $(\Pi_2)^{\{p,q\}}_\bot$ is

$$\bot \leftarrow \bot \qquad \bot \leftarrow not\ r \qquad r \leftarrow not\ p,$$

and we can check that $\{p, q\}$ does not satisfy the program, in accordance with Proposition 1.

Proposition 1 shows that the main theorem in Section 2.3 can be stated without mentioning external support: in conditions (b) and (c), we can replace "Y is externally supported by Π w.r.t. X" with "X does not satisfy $\Pi^Y_\bot$." This fact can be used to generalize the main theorem to arbitrary programs.

First we need to generalize the definition of a positive dependency graph to programs with nested expressions. The *positive dependency graph* of a program Π with nested expressions is the directed graph G such that

- its vertices are the literals occurring in Π, and
- its edges go from each literal in $poslit(Head)$ to each literal in $poslit(Body)$ for each rule $Head \leftarrow Body$ of Π.

If Π is canonical, this definition reduces to the earlier definition. The positive dependency graph of Π_5 is the same as the positive dependency graph of Π_2 as shown in Figure 1.

Once we define a positive dependency graph, the definition of a loop given in Section 2.3 remains the same for programs with nested expressions.

Main Theorem For any program Π with nested expressions and any consistent set X of literals, the following conditions are equivalent:

(a) X is an answer set for Π.

(b) X satisfies Π, and, for every set Y of literals that has a common element with X, X does not satisfy $\Pi^Y_\perp$.

(c) X satisfies Π, and, for every loop L of Π that is contained in X, X does not satisfy $\Pi^L_\perp$.

The translations T_b through T_e can be extended to finite programs Π with nested expressions that do not contain classical negation. For instance, the extended theory $T_b(\Pi)$ consists of (i) the implications *Body* $\supset$ *Head* for all rules *Head* $\leftarrow$ *Body* in Π, and (ii) the implications $\bigvee_{p\in Y} p \supset \neg\Pi^Y_\perp$ for all sets Y of atoms that occur in Π.

6 Conclusion

The following are the main contributions of this paper:

- We reformulated the definition of a loop formula so that loop formulas became a generalization of completion.
- We generalized the definition of a loop formula and the Lin/Zhao theorem to programs with nested expressions, not necessarily finite.
- We presented a model-theoretic account of loop formulas—the concept of an externally supported set—and showed that loop formulas are related to assumption sets and to unfounded sets.

It is interesting to note that the computational methods used in DLV and in SAT-based answer set solvers are related to each other, in view of our main theorem: DLV uses condition (d) [Leone *et al.*, 1997, Theorem 4.6] for finding answer sets for disjunctive programs, and ASSAT and CMODELS use condition (e) for nondisjunctive programs.

Acknowledgements

I am grateful to Vladimir Lifschitz and Fangzhen Lin for many useful discussions. Vladimir helped me develop the idea and improve the presentation significantly. I am also grateful to anonymous referees for their comments. Wolfgang Faber suggested that there may be a relationship between loop formulas and unfounded sets. Nicola Leone gave pointers to earlier work on unfounded sets and on assumption sets. This work was partially supported by NSF under Grant IIS-0412907.

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.

[Baral and Gelfond, 1994] Chitta Baral and Michael Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19,20:73–148, 1994.

[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.

[Erdem and Lifschitz, 2003] Esra Erdem and Vladimir Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, 3:499–518, 2003.

[Erdoğan and Lifschitz, 2004] Selim Erdoğan and Vladimir Lifschitz. Definitions in answer set programming. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 114–126, 2004.

[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.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080, 1988.

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

[Giunchiglia *et al.*, 2004] Enrico Giunchiglia, Yuliya Lierler, and Marco Maratea. SAT-based answer set programming. In *Proc. AAAI-04*, pages 61–66, 2004.

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

[Lee and Lifschitz, 2003] Joohyung Lee and Vladimir Lifschitz. Loop formulas for disjunctive logic programs. In *Proc. ICLP-03*, pages 451–465, 2003.

[Leone *et al.*, 1997] Nicola Leone, Pasquale Rullo, and Francesco Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Information and Computation*, 135(2):69–112, 1997.

[Lifschitz and Razborov, 2004] Vladimir Lifschitz and Alexander Razborov. Why are there so many loop formulas? *ACM Transactions on Computational Logic*, 2004. To appear.

[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.

[Lin and Zhao, 2004] Fangzhen Lin and Yuting Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157:115–137, 2004.

[Przymusinski, 1989] Teodor Przymusinski. On the declarative and procedural semantics of logic programs. *Journal of Automated Reasoning*, 5:167–205, 1989.

[Saccá and Zaniolo, 1990] Domenico Saccá and Carlo Zaniolo. Stable models and non-determinism in logic programs with negation. In *Proceedings of Symposium on Principles of Database Systems (PODS)*, pages 205–217, 1990.

[Van Gelder *et al.*, 1991] Allen Van Gelder, Kenneth Ross, and John Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650, 1991.