A Generalization of the Lin-Zhao Theorem

Paolo Ferraris
University of Texas at Austin, Texas, USA

Joohyung Lee
Arizona State University, Tempe, Arizona, USA

Vladimir Lifschitz
University of Texas at Austin, Texas, USA

Abstract

The theorem on loop formulas due to Fangzhen Lin and Yuting Zhao shows how to turn a logic program into a propositional formula that describes the program's stable models. In this paper we simplify and generalize the statement of this theorem. The simplification is achieved by modifying the definition of a loop in such a way that a program is turned into the corresponding propositional formula by adding loop formulas directly to the conjunction of its rules, without the intermediate step of forming the program's completion. The generalization makes the idea of a loop formula applicable to stable models in the sense of a very general definition that covers disjunctive programs, programs with nested expressions, and more.

1 Introduction

The theorem on loop formulas due to Fangzhen Lin and Yuting Zhao [Lin and Zhao, 2004] is an important result in the theory of stable models. It shows how to turn a logic program Π into a propositional formula that describes the stable models of Π. The reduction of the problem of computing stable models to the satisfiability problem for propositional formulas given by the Lin-Zhao theorem has led to the development of the answer set solvers ASSAT[1] and CMODELS[2]. If the program Π is tight [Fages, 1994, Erdem and Lifschitz, 2003] then the corresponding propositional formula is simply the completion of Π in the sense of [Clark, 1978]; otherwise the corresponding formula is the conjunction of the completion of Π with the additional formulas that Lin and Zhao called the "loop formulas" of Π. The number of loop formulas is exponential in the size of Π in the worst case, and there are reasons for this in complexity theory [Lifschitz and Razborov, 2006]. But in many cases the Lin-Zhao translation of Π into propositional logic is not much bigger than Π.

In this paper we show how the statement of the Lin-Zhao theorem can be simplified and generalized. The simplification is achieved by modifying the definition of a loop from [Lin and Zhao, 2004] in such a way that a program is turned into the corresponding propositional formula by adding loop formulas directly to the conjunction of its rules, without the intermediate step of forming the program's completion.

The generalization, on the other hand, makes the idea of a loop formula applicable to stable models in the sense of the very general definition proposed in [Ferraris, 2005] and [Ferraris and Lifschitz, 2005], which is essentially a reformulation of equilibrium logic [Pearce, 1997]. That general definition covers, in particular, disjunctive programs; the possibility of extending the Lin-Zhao theorem to the disjunctive case has been used to design a version of CMODELS that can handle disjunctive programs [Lierler, 2005]. The definition covers even arbitrary programs with nested expressions in the sense of [Lifschitz *et al.*, 1999], and more. The discussion of the semantics of aggregates (in particular, weight constraints with negative weights) in [Ferraris, 2005] shows that this high degree of generality is useful in some applications to knowledge representation.

Our version of the Lin-Zhao theorem is also more general than its original statement in another sense: it shows that loop formulas can be formed in two ways—not only "disjunctively" as in [Lin and Zhao, 2004], but also "conjunctively."

It can be viewed as an enhancement of the encoding of equilibrium logic by quantified propositional formulas proposed by David Pearce, Hans Tompits and Stefan Woltran [Pearce *et al.*, 2001]. If we eliminate quantifiers from that encoding, the result will be similar to the conjunction of loop formulas, but it will be much longer in many cases.

This paper is organized as follows. In Section 2 we discuss our generaliza-

[1] `http://assat.cs.ust.hk/` .

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

tion of the Lin-Zhao theorem for the simple case of "traditional" programs from [Gelfond and Lifschitz, 1988], and show how to extend it to disjunctive programs. The main theorem in full generality is stated in Section 3 and proved in Section 4. To make the paper self-contained, we have also included a review of the necessary background material from [Ferraris and Lifschitz, 2005] (Appendix A) and from [Pearce *et al.*, 2001] (Appendix B).

Preliminary reports on some of the work presented below are published in [Lee, 2005] and [Lee and Lifschitz, 2003].

2 Special Cases

2.1 Syntax and Semantics of Traditional Programs

A *traditional rule* is an expression of the form

$$a_1 \leftarrow a_2, \dots, a_m, \mathit{not}\ a_{m+1}, \dots, \mathit{not}\ a_n \tag{1}$$

where $n \geq m \geq 1$ and $a_1, \dots, a_n$ are propositional atoms. A *traditional program* is a finite set of traditional rules. We will identify a traditional rule (1) with the propositional formula

$$(a_2 \wedge \dots \wedge a_m \wedge \neg a_{m+1} \wedge \dots \wedge \neg a_n) \rightarrow a_1. \tag{2}$$

A traditional program Π will be identified with the conjunction of the formulas (2) corresponding to the rules of Π. In view of this convention, the definition of a stable model of a propositional formula from [Ferraris, 2005] and [Ferraris and Lifschitz, 2005], reproduced here in Appendix A, is applicable, in particular, to traditional programs; according to [Ferraris and Lifschitz, 2005, Proposition 28], it is equivalent in this special case to the familiar definition of a stable model proposed in [Gelfond and Lifschitz, 1988].

For example, the traditional program

$$\begin{array}{l} p \leftarrow q \\ q \leftarrow p \\ p \leftarrow \mathit{not}\ r \\ r \leftarrow \mathit{not}\ p \end{array} \tag{3}$$

can be viewed as alternative notation for the formula

$$(q \rightarrow p) \wedge (p \rightarrow q) \wedge (\neg r \rightarrow p) \wedge (\neg p \rightarrow r). \tag{4}$$

The stable models of this program are $\{p, q\}$ and $\{r\}$ (see Appendix A for the verification of a part of this claim).

$$\begin{tikzcd} p \arrow[r, bend left] & q \arrow[l, bend left] & r \end{tikzcd}$$

Figure 1: The dependency graph of program (3)

2.2 Main Theorem for Traditional Programs

The *(positive) dependency graph* of a traditional program Π is the directed graph such that

- its vertices are the atoms occurring in Π, and
- its edges go from a_1 to $a_2, \dots a_m$ for all rules (1) of Π.

A nonempty set L of atoms is called a *loop* of Π if, for every pair p, q of atoms in L, there exists a path (possibly of length 0) from p to q in the dependency graph of Π such that all vertices in this path belong to L. In other words, L is a loop of Π iff the subgraph of the dependency graph of Π induced by L is strongly connected. It is clear that any set consisting of a single atom is a loop.

For example, the dependency graph of program (3) is shown in Figure 1. This program has four loops:

$$\{p\},\ \{q\},\ \{r\},\ \{p,q\}. \tag{5}$$

Our definition of a loop is slightly different from the definition given in [Lin and Zhao, 2004], because it takes into account paths of length 0. This is what allows us to drop the completion step from the statement of the Lin-Zhao theorem; see Section 2.3 for details.

For any finite set Y of formulas, by $Y^\wedge$ and $Y^\vee$ we denote the conjunction and, respectively, disjunction of the elements of Y. Using this notation, we can write (2) as

$$(B^\wedge \wedge N) \to a_1 \tag{6}$$

where B is the set $\{a_2 \dots, a_m\}$ of "positive body atoms," and N is the "negative part" $\neg a_{m+1} \wedge \dots \wedge \neg a_n$.

For any set Y of atoms that occur in Π, the *external support formula* of Y, denoted by $ES_\Pi(Y)$, is the disjunction of the bodies $B^\wedge \wedge N$ of all rules (6) of Π such that

- $a_1 \in Y$ and
- $B \cap Y = \emptyset$.

The first condition expresses that the atom "supported" by (6) is an element of Y. The second condition expresses that this support is "external": the atoms B that it relies on do not belong to Y.

For instance, let Π be program (3), and let Y be $\{p, q\}$. Elements of Y are "supported" by each of the first three rules of (3), but in the case of the first two rules the support is not "external." Accordingly, the external support formula of $\{p, q\}$ is the body of the third rule, $\neg r$.

Main Theorem for Traditional Programs *Let Π be a traditional program, and let X be a set of atoms occurring in Π. If X is a model of Π then the following conditions are equivalent:*

(a) X is stable;

(b) for every set Y of atoms occurring in Π, X satisfies

$$Y^{\vee} \rightarrow ES_{\Pi}(Y); \tag{7}$$

(c) for every loop Y of Π, X satisfies (7);

(d) for every nonempty set Y of atoms occurring in Π, X satisfies

$$Y^{\wedge} \rightarrow ES_{\Pi}(Y); \tag{8}$$

(e) for every loop Y of Π, X satisfies (8).

We call (7) the *disjunctive loop formula* of Π corresponding to the set Y of atoms, and (8) its *conjunctive loop formula* for Y. The two formulas coincide when Y is a singleton.

For example, the loop formulas of program (3) are shown in Figure 2. According to the theorem above, a model of (3) is stable iff it satisfies each of the 8 disjunctive loop formulas. We can also say that a model of (3) is stable iff it satisfies the disjunctive loop formulas corresponding to the program's loops (5):

$$\begin{array}{rcl} p & \rightarrow & (q \vee \neg r) \\ q & \rightarrow & p \\ r & \rightarrow & \neg p \\ (p \vee q) & \rightarrow & \neg r. \end{array} \tag{9}$$

Alternatively, the stable models of (3) can be characterized as the models of (3) that satisfy the 7 conjunctive loop formulas shown in Figure 2, and, equivalently, as the models of (3) that satisfy the 4 conjunctive loop formulas corresponding to the program's loops.

Y	Disjunctive loop formula	Conjunctive loop formula
$\emptyset$	$\bot \rightarrow \bot$	
$\{p\}$	$p \rightarrow (q \vee \neg r)$	$p \rightarrow (q \vee \neg r)$
$\{q\}$	$q \rightarrow p$	$q \rightarrow p$
$\{r\}$	$r \rightarrow \neg p$	$r \rightarrow \neg p$
$\{p, q\}$	$(p \vee q) \rightarrow \neg r$	$(p \wedge q) \rightarrow \neg r$
$\{p, r\}$	$(p \vee r) \rightarrow (q \vee \neg r \vee \neg p)$	$(p \wedge r) \rightarrow (q \vee \neg r \vee \neg p)$
$\{q, r\}$	$(q \vee r) \rightarrow (p \vee \neg p)$	$(q \wedge r) \rightarrow (p \vee \neg p)$
$\{p, q, r\}$	$(p \vee q \vee r) \rightarrow (\neg r \vee \neg p)$	$(p \wedge q \wedge r) \rightarrow (\neg r \vee \neg p)$

Figure 2: The loop formulas of program (3)

Some of the implications between conditions (a)–(e) are obvious: it is easy to see that (b) implies both (c) and (d), and each of these two conditions implies (e). In Section 2.3 we show that the equivalence between conditions (a) and (c) is essentially a reformulation of the Lin-Zhao theorem. The equivalence between (a) and (d) is a reformulation of another published result; this is discussed in Section 2.5.

2.3 Comparison with the Lin-Zhao Theorem

We will now compare the theorem stated above with Theorem 1 from [Lin and Zhao, 2004]. The discussion here does not cover constraints (rules with empty heads), which are allowed by Lin and Zhao but are not allowed in traditional programs.

The *completion* of a traditional program Π is the set consisting of the equivalences

$$a_1 \leftrightarrow \bigvee (a_2 \wedge \cdots \wedge a_m \wedge \neg a_{m+1} \wedge \cdots \wedge \neg a_n) \tag{10}$$

for all atoms a_1 occurring in Π, where the disjunction extends over all rules (1) of Π with the head a_1. For instance, the completion of (3) is

$$\begin{array}{l} p \leftrightarrow (q \vee \neg r) \\ q \leftrightarrow p \\ r \leftrightarrow \neg p. \end{array} \tag{11}$$

We say that a loop L of a traditional program Π is *trivial* if

- L is a singleton, and

- the dependency graph of Π does not contain an edge from the element of L to itself.

For instance, the loops $\{p\}$, $\{q\}$, $\{r\}$ of program (3) are trivial; $\{p, q\}$ is the only nontrivial loop. If we add the rule $r \leftarrow r$ to program (3) then the loop $\{r\}$ will become nontrivial. Nontrivial loops in the sense of this definition are loops in the sense of [Lin and Zhao, 2004].

Lin-Zhao Theorem *For any traditional program Π and any set X of atoms occurring in Π, X is a stable model of Π iff X satisfies*

(i) the completion of Π, and

(ii) the disjunctive loop formulas for all nontrivial loops of Π.

For instance, the stable models $\{p, q\}$ and $\{r\}$ of (3) can be characterized as the models of (11) that satisfy the last of the formulas (9).

The part of the theorem from Section 2.1 that asserts the equivalence between conditions (a) and (c) is similar to the Lin-Zhao theorem. The difference is that the former does not refer to completion, and the latter does not refer to loop formulas for trivial loops.

It is not difficult to explain, however, why the set of formulas (i) and (ii) above is equivalent to the union of Π with the set of the disjunctive loop formulas of Π for all loops, both trivial and nontrivial. Indeed, (i) can be equivalently rewritten as the set of implications that consists of

(i′) the right-to-left implications from (10),

(i″) the left-to-right implications from (10) for the atoms a_1 such that the loop $\{a_1\}$ is trivial, and

(i‴) the left-to-right implications from (10) for the atoms a_1 such that the loop $\{a_1\}$ is nontrivial.

Group (i′) is equivalent to Π. Each implication

$$a_1 \rightarrow \bigvee (a_2 \wedge \cdots \wedge a_m \wedge \neg a_{m+1} \wedge \cdots \wedge \neg a_n) \tag{12}$$

in group (i″) is identical to the loop formula

$$\{a_1\}^{\vee} \rightarrow ES_{\Pi}(\{a_1\}), \tag{13}$$

because, for every rule (1) of Π with the head a_1,

$$B \cap \{a_1\} = \{a_2, \ldots, a_m\} \cap \{a_1\} = \emptyset.$$

Finally, group (i$'''$) can be dropped in the presence of (ii), because each implication (12) in group (i$'''$) is entailed by the corresponding loop formula (13): the loop formula can be obtained from (12) by dropping the disjunctive terms with $a_1 \in \{a_2, \dots, a_m\}$.

2.4 Extension to Disjunctive Programs

As an intermediate step before discussing the main theorem in full generality, we will consider the special case of "disjunctive" programs. Disjunctive rules are often defined as expressions of the form

$$a_1; \dots; a_k \leftarrow a_{k+1}, \dots, a_m, \mathit{not}\ a_{m+1}, \dots, \mathit{not}\ a_n \tag{14}$$

($n \geq m \geq k \geq 0$), and the definition of a stable model from Appendix A can be applied to finite sets of such rules if we treat (14) as alternative notation for the formula

$$(a_{k+1} \wedge \cdots \wedge a_m \wedge \neg a_{m+1} \wedge \cdots \wedge \neg a_n) \to (a_1 \vee \cdots \vee a_k). \tag{15}$$

The understanding of disjunctive rules in this section will be more general. We say that a propositional formula is *negative* if every occurrence of every atom in this formula is in the scope of a negation or in the antecedent of an implication. For instance, the conjunction $\neg a_{m+1} \wedge \cdots \wedge \neg a_n$ in (15) is negative; any formula of the form $F \to \neg G$ is negative. The 0-place connectives $\top$ and $\bot$ are negative formulas also, because they don't contain atoms. A *disjunctive rule* is a formula of the form

$$(B^{\wedge} \wedge N) \to A^{\vee} \tag{16}$$

where A and B are finite sets of atoms, and N is a negative formula. For instance,

$$(p \wedge \neg(q \wedge \neg r)) \to s$$

is a disjunctive rule in the sense of this definition; in the language of LPARSE,[3] it can be written as

```
s :- p, {q, not r}1.
```

A *disjunctive program* is a conjunction of (0 or more) disjunctive rules.

The definition of the dependency graph (Section 2.2) is extended to disjunctive programs in a straightforward way: the vertices of the graph are the atoms occurring in the program, and its edges go from the elements of A

[3] `http://www.tcs.hut.fi/Software/smodels/lparse.ps` .

to the elements of B for all rules (16) of the program. For instance, a rule of the form

$$(b_1 \wedge N) \rightarrow (a_1 \vee a_2)$$

contributes two edges to the dependency graph: from a_1 to b_1 and from a_2 to b_1. Constraints ($A = \emptyset$) and rules with a negative body ($B = \emptyset$) don't contribute edges to the dependency graph.

The definition of a loop in terms of the dependency graph remains the same as in Section 2.2.

For any set Y of atoms that occur in a disjunctive program Π, the *external support formula* of Y, denoted by $ES_\Pi(Y)$, is the disjunction of the formulas

$$B^\wedge \wedge N \wedge \bigwedge_{a \in A \setminus Y} \neg a \tag{17}$$

for all rules (16) of Π such that

- $A \cap Y \neq \emptyset$, and
- $B \cap Y = \emptyset$.

When Π is a traditional program, this definition reduces to the definition of ES_Π given in Section 2.2.

The theorem from Section 2.2 remains correct after replacing "traditional program" in its statement with "disjunctive program." The terms "disjunctive loop formula" and "conjunctive loop formula" will be applied to formulas (7) and (8) when Π is an arbitrary disjunctive program.

For instance, consider the program

$$\begin{array}{l} p\,; s \leftarrow q \\ q \leftarrow p \\ p\,; r \leftarrow \mathit{not}\ s \end{array}$$

which is the "logic programming representation" of the formula

$$(q \rightarrow (p \vee s)) \wedge (p \rightarrow q) \wedge (\neg s \rightarrow (p \vee r)). \tag{18}$$

The loops of this program are

$$\{p\},\ \{q\},\ \{r\},\ \{s\},\ \{p, q\},$$

and the corresponding disjunctive loop formulas are

$$\begin{array}{rcl} p & \rightarrow & ((q \wedge \neg s) \vee (\neg s \wedge \neg r)) \\ q & \rightarrow & p \\ r & \rightarrow & (\neg s \wedge \neg p) \\ s & \rightarrow & (q \wedge \neg p) \\ (p \vee q) & \rightarrow & (\neg s \wedge \neg r). \end{array} \tag{19}$$

The stable models $\{p, q\}$, $\{r\}$ of (18) can be characterized as the models of (18) that satisfy (19).

2.5 Relation to Unfounded Sets

For programs consisting of rules of the form (14), the equivalence between conditions (a) and (d) from the statement of the main theorem has been established earlier, in a somewhat different form. Saccá and Zaniolo [1990] showed that the stable models of what we call here traditional programs can be characterized in terms of "unfounded sets."[4] Leone *et al.* [1997] extended the notion of an unfounded set and the theorem by Saccá and Zaniolo to disjunctive rules of the form (14).

Their definition can be further extended to arbitrary disjunctive programs in the sense of this section as follows. A set Y of atoms is *unfounded* by a disjunctive program Π w.r.t. a set X of atoms if, for each rule (16) of Π such that $A \cap Y \neq \emptyset$,

- $X \not\models B^{\wedge} \wedge N$, or
- $B \cap Y \neq \emptyset$, or
- $X \cap (A \setminus Y) \neq \emptyset$.

It is easy to see that $X \models ES_{\Pi}(Y)$ iff Y is not unfounded by Π w.r.t. X.

A set X of atoms is called *unfounded-free* if it has no nonempty subsets unfounded w.r.t. X. The equivalence between conditions (a) and (d) can be reformulated as follows: *for any model X of a disjunctive program Π, X is stable iff X is unfounded-free.* This is a generalization of Corollary 2 from [Saccá and Zaniolo, 1990], and of Theorem 4.6 from [Leone *et al.*, 1997].

3 General Theory of Loop Formulas

Our goal now is to extend the definition of a loop and the definition of a loop formula, stated above for traditional programs (Section 2.2) and for disjunctive programs (Section 2.4), to the general case of arbitrary propositional formulas, and to state the main theorem in full generality.

[4] Their theorem refers actually to "assumption sets" rather than unfounded sets. But as the authors noted, in the context of this theorem the two concepts are equivalent. Unfounded sets were originally introduced for the purpose of characterizing the negative consequences of a program under the well-founded semantics [Van Gelder *et al.*, 1991].

For simplicity, we assume here that the only propositional connectives allowed in formulas are

$$\bot,\ \wedge,\ \vee \text{ and } \rightarrow,$$

and all other connectives are treated as abbreviations, as in Section B.3. For instance, (2) is now viewed as an abbreviation for

$$(a_2 \wedge \cdots \wedge a_m \wedge (a_{m+1} \rightarrow \bot) \wedge \cdots \wedge (a_n \rightarrow \bot)) \rightarrow a_1. \tag{20}$$

Under this simplifying assumption, the definition of a negative formula from Section 2.4 can be stated as follows: a formula is *negative* if every occurrence of every atom in this formula belongs to the antecedent of an implication.

3.1 Loops

An occurrence of a formula G in a formula F is *positive* if the number of implications in F containing that occurrence in the antecedent is even; it is *strictly positive* if that number is 0.[5] In (20), for instance, the occurrences of $a_1, a_{m+1}, \ldots, a_n$ are positive, but only the first of them is strictly positive. It is clear that a formula F is negative iff it has no strictly positive occurrences of atoms.

Note that we apply the term "negative" to formulas, and the terms "positive" and "strictly positive" to occurrences of one formula in another.

We say that an atom a *depends* on an atom b in an implication $G \rightarrow H$ if

- a has a strictly positive occurrence in H, and
- b has a positive occurrence in G that does not belong to any occurrence of a negative formula in G.

The *dependency graph* of a formula F is the directed graph such that

- its vertices are the atoms that occur in F, and
- it has an edge from a vertex a to a vertex b if a depends on b in an implication that has a strictly positive occurrence in F.

In application to traditional programs, the new definition of the dependency graph is equivalent to the definition from Section 2.2. Indeed, assume that F is a conjunction of formulas of the form (20). Implications occurring

[5]The concept of a strictly positive occurrence plays an important role in intuitionistic logic; see, for instance, [Troelstra and Schwichtenberg, 1996, Theorem 4.2.3].

in F are of two kinds: conjunctive terms (20) and implications of the form $a_i \rightarrow \bot$. The edges contributed to the dependency graph by (20) go from a_1 to $a_2, \dots, a_m$. Implications of the form $a_i \rightarrow \bot$ do not contribute edges to the dependency graph.

More generally, in application to disjunctive programs the new definition of the dependency graph is equivalent to the definition from Section 2.4. Indeed, assume that F is a conjunction of formulas of the form (16). Implications occurring in F are of two kinds: conjunctive terms (16) and implications that are subformulas of N in one of these conjunctive terms. The edges contributed to the dependency graph by the implications (16) go from elements of A to elements of B; these implications do not contribute any other edges, because N is negative. Implications from N do not contribute edges to the dependency graph: if an implication $G \rightarrow H$ has a strictly positive occurrence in a negative formula N then H is a negative formula also, and no occurrence of an atom in H can be strictly positive.

Consider now some formulas other than disjunctive programs. Formula

$$(p \rightarrow q) \vee r \tag{21}$$

is a disjunction of two traditional rules. Its dependency graph has one edge, from q to p. The dependency graph of the nested implication

$$((p \rightarrow q) \rightarrow r) \rightarrow s$$

has two edges—from s to r and from s to p. The dependency graph of

$$((p \rightarrow \neg q) \rightarrow r) \rightarrow s$$

has only one edge, from s to r, because the formula $p \rightarrow \neg q$ is negative.

Given this definition of a dependency graph, loops are defined in the same way as in Section 2.2: a *loop* of a formula F is a nonempty set of atoms occurring in F such that the subgraph of the dependency graph of F induced by that set is strongly connected.

3.2 Loop Formulas

For any set Y of atoms occurring in a formula F, we want to define a formula that would be similar to the external support formula $ES_F(Y)$ in the special case when F is a disjunctive program. It is easier to define a formula such that its *negation* is similar to $ES_F(Y)$.

Such a formula $NES_F(Y)$ is defined recursively, as follows:

- for an atom a, $NES_a(Y)$ is $\bot$ if $a \in Y$, and a otherwise;
- $NES_\bot(Y) = \bot$;
- $NES_{F \wedge G}(Y) = NES_F(Y) \wedge NES_G(Y)$;
- $NES_{F \vee G}(Y) = NES_F(Y) \vee NES_G(Y)$;
- $NES_{F \rightarrow G}(Y) = (NES_F(Y) \rightarrow NES_G(Y)) \wedge (F \rightarrow G)$.

For instance, if F is $p \rightarrow q$ then

$$\begin{array}{rcl} NES_F(\{q\}) & = & (NES_p(\{q\}) \rightarrow NES_q(\{q\})) \wedge (p \rightarrow q) \\ & = & (p \rightarrow \bot) \wedge (p \rightarrow q) \\ & \leftrightarrow & \neg p. \end{array}$$

The definitions of *ES* and *NES* look very different from each other. But the calculation above shows that in the case of $p \rightarrow q$ the formula $NES_F(\{q\})$ is equivalent to the negation of the external support formula p of $\{q\}$. The following proposition shows that $NES_\Pi(Y)$ is "almost equivalent" to the negation of $ES_\Pi(Y)$ for any disjunctive program Π:

Theorem 1 *If X is a model of a disjunctive program Π then, for any set Y of atoms,*

$$X \models NES_\Pi(Y) \text{ iff } X \models \neg ES_\Pi(Y).$$

This fact suggests that $\neg NES_F(Y)$ may be an acceptable counterpart of the external support formula of Y when F is syntactically different from disjunctive programs. The main theorem, stated in the next section, shows that this is indeed the case. Its statement refers to the formulas

$$Y^\vee \rightarrow \neg NES_F(Y) \tag{22}$$

and

$$Y^\wedge \rightarrow \neg NES_F(Y), \tag{23}$$

which can be called the (disjunctive and conjunctive) *loop formulas* of a formula F corresponding to the set Y of atoms.

3.3 Main Theorem

Theorem 2 (Main Theorem) *Let F be a propositional formula, and let X be a set of atoms occurring in F. If X is a model of F then the following conditions are equivalent:*

(a) X is stable;

(b) for every set Y of atoms occurring in F, X satisfies (22);

(c) for every loop Y of F, X satisfies (22);

(d) for every nonempty set Y of atoms occurring in F, X satisfies (23);

(e) for every loop Y of F, X satisfies (23).

Theorem 1 shows that the theorem stated in Section 2.2 and its extension to disjunctive programs (Section 2.4) can be viewed as special cases of Theorem 2.

As an example, let's apply Theorem 2 to formula (21). Its loops are the singletons $\{p\}$, $\{q\}$, $\{r\}$, and the corresponding loop formulas (22) are

$$\begin{array}{l} p \to \neg(((\bot \to q) \wedge (p \to q)) \vee r), \\ q \to \neg(((p \to \bot) \wedge (p \to q)) \vee r), \\ r \to \neg(((p \to q) \wedge (p \to q)) \vee \bot). \end{array}$$

The conjunction of these formulas is equivalent to

$$\neg q \wedge \neg r. \tag{24}$$

According to the main theorem, the stable models of (21) can be characterized as the sets that satisfy both (21) and (24). The conjunction of (21) with (24) is equivalent to $\neg p \wedge \neg q \wedge \neg r$, so that the only stable model of (21) is $\emptyset$.

4 Proofs

4.1 Proof of Theorem 1

Lemma 1 *For any formula F and any set Y of atoms,*

(a) $NES_F(Y)$ entails F;

(b) if F has no strictly positive occurrences of atoms from Y then $NES_F(Y)$ is equivalent to F.

Proof. (a) by induction on F. (b) by induction on F; consider the case when F is $G \to H$. By (a), $NES_G(Y)$ entails G; by the induction hypothesis, $NES_H(Y)$ is equivalent to H. Consequently,

$$\begin{array}{rcl} NES_F(Y) & = & (NES_G(Y) \to NES_H(Y)) \wedge (G \to H) \\ & \leftrightarrow & (NES_G(Y) \to H) \wedge (G \to H) \\ & \leftrightarrow & ((NES_G(Y) \vee G) \to H) \\ & \leftrightarrow & (G \to H) \\ & = & F. \end{array}$$

∎

Theorem 1 *If X is a model of a disjunctive program Π then, for any set Y of atoms,*

$$X \models NES_\Pi(Y) \text{ iff } X \models \neg ES_\Pi(Y).$$

Proof. Since $NES_\Pi(Y)$ is the conjunction of the formulas $NES_R(Y)$ for all rules R of Π, and $ES_\Pi(Y)$ is the disjunction of the formulas $ES_R(Y)$, it is sufficient to consider the case when Π is a single rule (16). In this case, $\neg ES_\Pi(Y)$ is equivalent to

$$(B^\wedge \wedge N) \to (A \setminus Y)^\vee \tag{25}$$

if $A \cap Y \neq \emptyset$ and $B \cap Y = \emptyset$, and is $\top$ otherwise. We need to show, assuming (16), that this formula is equivalent to $NES_\Pi(Y)$. In the presence of (16), using Lemma 1(a),

$$\begin{array}{rcl} NES_\Pi(Y) & = & (NES_{B^\wedge \wedge N}(Y) \to NES_{A^\vee}(Y)) \wedge ((B^\wedge \wedge N) \to A^\vee) \\ & \leftrightarrow & NES_{B^\wedge \wedge N}(Y) \to NES_{A^\vee}(Y) \\ & \leftrightarrow & (NES_{B^\wedge}(Y) \wedge NES_N(Y)) \to NES_{A^\vee}(Y) \\ & \leftrightarrow & (NES_{B^\wedge}(Y) \wedge N) \to NES_{A^\vee}(Y) \\ & \leftrightarrow & (NES_{B^\wedge}(Y) \wedge N) \to (A \setminus Y)^\vee. \end{array}$$

If $B \cap Y \neq \emptyset$ then the last formula contains the conjunctive term $\bot$ in the antecedent, and consequently is equivalent to $\top$. Otherwise, it can be rewritten as (25). It remains to note that if $A \cap Y = \emptyset$ then (25) is identical to the assumption (16) and consequently can be rewritten as $\top$. ∎

4.2 Proof of Theorem 2: Equivalence of (a), (b), (d)

In the following lemma, F is a propositional formula, and $\mathbf{a}$ is a list of distinct atoms $a_1, \dots, a_n$ containing all atoms occurring in F. For the definitions of $F^*(\mathbf{v})$ and $\overrightarrow{Y}$, see Section B.3.

Lemma 2 *For any sets X, Y of atoms, $X \models NES_F(Y)$ iff $X \models F^*(\overrightarrow{X \setminus Y})$.*

Proof. By induction on F. Consider the case when F is an atom. If $F \in Y$ then each of the formulas $NES_F(Y)$, $F^*(\overrightarrow{X \setminus Y})$ is $\bot$. Otherwise $NES_F(Y)$ is F, while $F^*(\overrightarrow{X \setminus Y})$ is $\top$ or $\bot$ depending on whether $F \in X$. The other cases are straightforward. ∎

Proof of the equivalence of conditions (a), (b), (d) in the statement of Theorem 2. Let $\mathbf{a}$ be the list of atoms occurring in F, and let a subset X of $\mathbf{a}$ be a model of F. By the Pearce-Tompits-Woltran theorem, and in view of the fact that PTW$[F]$ can be written in the form (32), condition (a) ("X is stable") is equivalent to

$$X \models \bigwedge_{Y \subseteq \mathbf{a}} (\overrightarrow{Y} < \mathbf{a} \rightarrow \neg F^*(\overrightarrow{Y}))$$

and consequently to

$$X \models \bigwedge_{Y \subset X} \neg F^*(\overrightarrow{Y}).$$

Using Lemma 2, we can show that this condition is equivalent to condition (b):

$$\begin{aligned}
X \models \bigwedge_{Y \subset X} \neg F^*(\overrightarrow{Y}) \quad & \text{iff} \quad X \models \bigwedge_{Z \subseteq \mathbf{a}:\ Z \cap X \neq \emptyset} \neg F^*(\overrightarrow{X \setminus Z}) \\
& \text{iff} \quad X \models \bigwedge_{Z \subseteq \mathbf{a}:\ Z \cap X \neq \emptyset} \neg NES_F(Z) \\
& \text{iff} \quad X \models \bigwedge_{Z \subseteq \mathbf{a}} (Z^{\vee} \rightarrow \neg NES_F(Z)) \\
& \text{iff} \quad X \models Z^{\vee} \rightarrow \neg NES_F(Z) \\
& \qquad \text{for all subsets } Z \text{ of } \mathbf{a}.
\end{aligned}$$

It is also equivalent to (d):

$$\begin{array}{rll}
X \models \bigwedge\limits_{Y \subset X} \neg F^*(\overrightarrow{Y}) & \text{iff} & X \models \bigwedge\limits_{Z \subseteq \mathbf{a}:\ Z \subseteq X,\ Z \neq \emptyset} \neg F^*(\overrightarrow{X \setminus Z}) \\
& \text{iff} & X \models \bigwedge\limits_{Z \subseteq \mathbf{a}:\ Z \subseteq X,\ Z \neq \emptyset} \neg NES_F(Z) \\
& \text{iff} & X \models \bigwedge\limits_{Z \subseteq \mathbf{a}:\ Z \neq \emptyset} (Z^\wedge \rightarrow \neg NES_F(Z)) \\
& \text{iff} & X \models Z^\wedge \rightarrow \neg NES_F(Z) \\
& & \quad \text{for all nonempty subsets } Z \text{ of } \mathbf{a}.
\end{array}$$

∎

4.3 Proof of Theorem 2: Equivalence of (c), (e) to the other conditions

Lemma 3 *For any formula F, set Y of atoms, and subset Z of Y,*

(a) if every positive occurrence of every atom from $Y \setminus Z$ in F belongs to a negative formula then $NES_F(Z)$ entails $NES_F(Y)$;

(b) if every nonpositive occurrence of every atom from $Y \setminus Z$ in F belongs to a negative formula then $NES_F(Y)$ entails $NES_F(Z)$.

Proof. Both parts are proved simultaneously by induction on F. Assume that F is an atom. (a) Since every positive occurrence of every atom from $Y \setminus Z$ in F belongs to a negative formula, $F \notin Y \setminus Z$, so that $NES_F(Z)$ is the same formula as $NES_F(Y)$. (b) Since Z is a subset of Y, $NES_F(Y)$ is equal to $NES_F(Z)$ or to $\bot$. The cases when F is $\bot$, a conjunction or a disjunction are straightforward. Assume that F is $G \rightarrow H$. If F is negative then, by Lemma 1(b), each of the formulas $NES_F(Y)$, $NES_F(Z)$ is equivalent to F. Assume that F is not negative. (a) Every nonpositive occurrence of every atom from $Y \setminus Z$ in G belongs to a negative formula, and so does every positive occurrence of every atom from $Y \setminus Z$ in H. We need to show that

$$(NES_G(Z) \rightarrow NES_H(Z)) \wedge (G \rightarrow H)$$

entails

$$(NES_G(Y) \rightarrow NES_H(Y)) \wedge (G \rightarrow H),$$

This is clear from the fact that, by the induction hypothesis, $NES_G(Y)$ entails $NES_G(Z)$ and $NES_H(Z)$ entails $NES_H(Y)$. (b) Similar. ∎

Lemma 4 *For any formula F and any nonempty set Y of atoms, there exists a subset Z of Y such that*

(a) Z is a loop of F, and

(b) the dependency graph of F has no edges from atoms in Z to atoms in $Y \setminus Z$.

Proof. Consider the strongly connected components of the subgraph of the dependency graph of F induced by Y. They form a finite acyclic graph. Any terminal vertex of that graph satisfies conditions (a) and (b). ∎

Lemma 5 *Let X be a model of a formula F, Y a set of atoms, and Z a nonempty subset of Y such that the dependency graph of F has no edges from atoms in Z to atoms in $Y \setminus Z$. If $X \models NES_F(Y)$ then $X \models NES_F(Z)$.*

Proof. By induction on F.

Case 1: F is an atom or $\bot$. Then the $NES_F(Y)$ is equal to $NES_F(Z)$ or to $\bot$.

Case 2: F is $G \wedge H$. All edges in the dependency graphs of G and H belong to the dependency graph of F, so that the inductive hypothesis can be applied both to G and to H.

Case 3: F is $G \vee H$. Similar to Case 2.

Case 4: F is $G \rightarrow H$. Assume that the dependency graph of F has no edges from Z to $Y \setminus Z$, and that X satisfies $NES_F(Y)$:

$$X \models (NES_G(Y) \rightarrow NES_H(Y)) \wedge (G \rightarrow H) \tag{26}$$

but doesn't satisfy $NES_F(Z)$:

$$X \not\models (NES_G(Z) \rightarrow NES_H(Z)) \wedge (G \rightarrow H). \tag{27}$$

Since X is a model of $G \rightarrow H$, X doesn't satisfy the first conjunctive term of (27), so that

$$X \models NES_G(Z) \tag{28}$$

and

$$X \not\models NES_H(Z). \tag{29}$$

By Lemma 1(a), (28) implies $X \models G$. Since X is a model of $G \rightarrow H$, it follows that $X \models H$. In combination with (29) and Lemma 1(b), this fact shows that H contains a strictly positive occurrence of an atom from Z. Since there are no edges from Z to $Y \setminus Z$ in the dependency graph of F, it follows that every positive occurrence of every atom from $Y \setminus Z$ in G belongs to a negative formula. By Lemma 3(a), we can conclude that $NES_G(Z)$ entails $NES_G(Y)$. Then, in view of (28), $X \models NES_G(Y)$. By (26), it follows that $X \models NES_H(Y)$. Since every edge in the dependency graph of H belongs to the dependency graph of F, the inductive hypothesis is applicable to H, and we can further conclude that $X \models NES_H(Z)$, which contradicts (29). ∎

Proof of the equivalence of conditions (d) and (e) in the statement of Theorem 2. Let X be a model of F. It is clear that (d) implies (e). Assume that (d) does not hold, and let Y be a nonempty set of atoms such that X does not satisfy loop formula (23), so that

$$X \models Y^{\wedge} \tag{30}$$

and

$$X \models NES_F(Y). \tag{31}$$

By Lemma 4, there exists a subset Z of Y such that Z is a loop of F, and the dependency graph of F has no edges from Z to $Y \setminus Z$. From (30) we conclude that $X \models Z^{\wedge}$. By Lemma 5, (31) implies that $X \models NES_F(Z)$. Consequently (e) does not hold either. ∎

Proof of the equivalence of condition (c) to the other conditions in the statement of Theorem 2. Clearly (b) implies (c), and (c) implies (e). On the other hand, we have already established that (b) is equivalent to (e). ∎

5 Conclusion

We modified the definition of a loop due to Lin and Zhao so that the reference to the program's completion in the statement of their theorem became unnecessary, and generalized the theorem, first to disjunctive programs, and then to arbitrary propositional formulas.

In the most general framework, the definition of the dependency graph is guided by three ideas. First, rules of a given program can be viewed as implications that occur in it strictly positively. Second, head atoms of a rule can be viewed as atoms that occur in its head strictly positively. Third, positive body atoms of a rule can be viewed as atoms that occur in its body positively and do not belong to any negative formula.

The most general definition of a loop formula, on the other hand, is motivated by a relationship between external support formulas and a syntactic transformation introduced by Pearce, Tompits and Woltran.

In this paper we did not discuss logic programs with two negations [Gelfond and Lifschitz, 1990], which are important in many applications to knowledge representation. Instead of treating the second negation as an additional syntactic construct, we can think of it in terms of distinguishing between atoms of two kinds, coming in "complementary pairs," and in terms of "coherent" stable models [Ferraris and Lifschitz, 2005, Section 3.9].

Acknowledgements

This research was partially supported by the National Science Foundation under Grant IIS-0412907.

A Definition of a Stable Model

Atoms and formulas are understood here as in propositional logic. As usual, we identify truth assignments with sets of atoms; for instance, the truth assignment that makes the atom p true and all other atoms false is identified with $\{p\}$. A *model* of a formula F is a set of atoms that satisfies F.

According to [Ferraris and Lifschitz, 2005, Section 2.1], the *reduct* F^X of a formula F relative to a set X of atoms is the formula obtained from F by replacing each maximal subformula that is not satisfied by X with $\bot$ ("false"). We say that X is a *stable model* (or an *answer set*) of F if X is minimal among the sets satisfying F^X. The minimality of X is understood here in the sense of set inclusion.

Clearly, every set that is a stable model of F according to this definition is a model of F. Indeed, if X does not satisfy F then F^X is $\bot$.

Thus we can verify that X is a stable model of F as follows:

(i) mark in F the maximal subformulas that are not satisfied by X;

(ii) replace each of these subformulas with $\bot$ (after that, equivalent transformations of classical propositional logic can be used to simplify the result);

(iii) check that the resulting formula is satisfied by X;

(iv) check that it is not satisfied by any proper subset of X.

For instance, to check that $\{r\}$ is a stable model of (4), we do the following:

(i) mark the maximal subformulas of (4) that are not satisfied by $\{r\}$:

$$(\underline{q} \rightarrow \underline{p}) \wedge (\underline{p} \rightarrow \underline{q}) \wedge (\underline{\neg r} \rightarrow \underline{p}) \wedge (\neg \underline{p} \rightarrow r);$$

(ii) replace these subformulas with $\bot$:

$$(\bot \rightarrow \bot) \wedge (\bot \rightarrow \bot) \wedge (\bot \rightarrow \bot) \wedge (\neg\bot \rightarrow r);$$

simplify:

$$r;$$

(iii) check that the last formula is satisfied by $\{r\}$;

(iv) check that it is not satisfied by $\emptyset$.

As another example, the model $\{r\}$ of formula (21) is *not* stable:

(i) mark the maximal subformulas of (21) that are not satisfied by $\{r\}$:

$$(\underline{p} \rightarrow \underline{q}) \vee r;$$

(ii) replace these subformulas with $\bot$:

$$(\bot \rightarrow \bot) \vee r;$$

simplify:

$$\top.$$

The last formula is satisfied by $\{r\}$, but it is also satisfied by the proper subset $\emptyset$ of $\{r\}$. In fact, the only stable model of (21) is $\emptyset$.

B Propositional Circumscription and the Pearce-Tompits-Woltran Theorem

The Pearce-Tompits-Woltran theorem is about a syntactic transformation that is similar to circumscription [McCarthy, 1980, McCarthy, 1986, Lifschitz, 1994]. For this reason, our review includes a brief discussion of that concept.

B.1 Second-Order Propositional Formulas

Second-order propositional formulas (also known as *quantified Boolean formulas*) are formed from propositional atoms (in this paper, $p, q, \dots$) and an infinite supply of propositional variables $(x, y, \dots)$ using propositional connectives and the quantifiers $\forall$, $\exists$. The usual recursive definition of satisfaction for propositional formulas is extended to second-order propositional formulas without free variables as follows: a truth assignment (or a set of atoms) satisfies $\forall v F(v)$ if it satisfies both $F(\bot)$ and $F(\top)$; it satisfies $\exists v F(v)$ if it satisfies at least one of these two formulas. A second-order propositional formula is *logically valid* if its universal closure is satisfied by all truth assignments.

Quantifiers can be eliminated from any second-order propositional formula by repeatedly replacing parts of the form $\forall v F(v)$ with $F(\bot) \wedge F(\top)$, and parts of the form $\exists v F(v)$ with $F(\bot) \vee F(\top)$. This transformation turns logically valid formulas without free variables into tautologies. For example,

$$\begin{array}{rcl}\forall x \exists y(y \leftrightarrow p \wedge x) & \leftrightarrow & \exists y(y \leftrightarrow p \wedge \top) \wedge \exists y(y \leftrightarrow p \wedge \bot)\\ & \leftrightarrow & \exists y(y \leftrightarrow p) \wedge \exists y \neg y\\ & \leftrightarrow & ((\top \leftrightarrow p) \vee (\bot \leftrightarrow p)) \wedge (\neg\top \vee \neg\bot)\\ & \leftrightarrow & \top.\end{array}$$

B.2 Propositional Circumscription

The review of circumscription in this section is limited to the propositional case of parallel circumscription with no varied constants.

Let $\mathbf{a}$ be a tuple of distinct atoms $a_1, \dots, a_n$, and $F(\mathbf{a})$ a propositional formula. The *circumscription of* $\mathbf{a}$ *in* $F(\mathbf{a})$, denoted by $\mathrm{CIRC}[F(\mathbf{a}); \mathbf{a}]$, is the second-order propositional formula

$$F(\mathbf{a}) \wedge \neg\exists \mathbf{v}(\mathbf{v} < \mathbf{a} \wedge F(\mathbf{v})),$$

where $\mathbf{v}$ is a tuple of n distinct propositional variables $v_1, \dots, v_n$, and $\mathbf{v} < \mathbf{a}$ stands for

$$(v_1 \rightarrow a_1) \wedge \cdots \wedge (v_n \rightarrow a_n) \wedge \neg((a_1 \rightarrow v_1) \wedge \cdots \wedge (a_n \rightarrow v_n)).$$

For instance,

$$\begin{array}{rcl}\mathrm{CIRC}[p \vee q; p] & = & (p \vee q) \wedge \neg\exists x(x < p \wedge (x \vee q))\\ & \leftrightarrow & (p \vee q) \wedge \neg((\bot < p \wedge (\bot \vee q)) \vee (\top < p \wedge (\top \vee q)))\\ & \leftrightarrow & (p \vee q) \wedge \neg((p \wedge q)) \vee (\bot \wedge \top))\\ & \leftrightarrow & (p \vee q) \wedge \neg(p \wedge q).\end{array}$$

B.3 Pearce-Tompits-Woltran Theorem

In this section we assume that the connectives used in propositional formulas are

$$\bot,\ \wedge,\ \vee \text{ and } \rightarrow;$$

$\top$ stands for $\bot \rightarrow \bot$, $\neg F$ for $F \rightarrow \bot$, and $F \leftrightarrow G$ for $(F \rightarrow G) \wedge (G \rightarrow F)$.

Let $a_1, \dots, a_n$ be all atoms occurring in a propositional formula F. By PTW$[F]$ we denote the second-order propositional formula

$$F \wedge \neg \exists \mathbf{v}(\mathbf{v} < \mathbf{a} \wedge F^*(\mathbf{v})),$$

where $\mathbf{a}$ stands for $a_1, \dots, a_n$, $\mathbf{v}$ is a tuple of n distinct propositional variables $v_1, \dots, v_n$, and $F^*(\mathbf{v})$ is defined recursively, as follows:

- $(a_i)^* = v_i$;
- $\bot^* = \bot$;
- $(F \wedge G)^* = F^* \wedge G^*$;
- $(F \vee G)^* = F^* \vee G^*$;
- $(F \rightarrow G)^* = (F^* \rightarrow G^*) \wedge (F \rightarrow G)$.

For instance, let F be the formula $p \wedge (p \rightarrow (q \vee r))$, corresponding to the disjunctive program

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

Then

$$\begin{array}{rcl} F^* & = & p^* \wedge (p \rightarrow (q \vee r))^* \\ & = & p^* \wedge (p^* \rightarrow (q \vee r)^*) \wedge (p \rightarrow (q \vee r)) \\ & = & p^* \wedge (p^* \rightarrow (q^* \vee r^*)) \wedge (p \rightarrow (q \vee r)) \\ & = & x \wedge (x \rightarrow (y \vee z)) \wedge (p \rightarrow (q \vee r)) \\ & \leftrightarrow & x \wedge (y \vee z) \wedge (p \rightarrow (q \vee r)) \end{array}$$

and

$$
\begin{aligned}
\mathrm{PTW}[F] \quad \leftrightarrow \quad & p \wedge (p \rightarrow (q \vee r)) \\
& \wedge \neg \exists xyz((x,y,z) < (p,q,r) \\
& \qquad\qquad \wedge x \wedge (y \vee z) \wedge (p \rightarrow (q \vee r))) \\
\leftrightarrow \quad & p \wedge (q \vee r) \\
& \wedge \neg \exists xyz((x,y,z) < (p,q,r) \wedge x \wedge (y \vee z)) \\
\leftrightarrow \quad & p \wedge (q \vee r) \wedge \neg \exists yz((\top, y, z) < (p,q,r) \wedge (y \vee z)) \\
\leftrightarrow \quad & p \wedge (q \vee r) \wedge \neg \exists yz((y,z) < (q,r) \wedge p \wedge (y \vee z)) \\
\leftrightarrow \quad & p \wedge (q \vee r) \wedge \neg \exists yz((y,z) < (q,r) \wedge (y \vee z)) \\
\leftrightarrow \quad & p \wedge (q \vee r) \wedge \neg \exists yz((\neg y \wedge z \wedge q \wedge r) \\
& \qquad\qquad \vee (y \wedge \neg z \wedge q \wedge r)) \\
\leftrightarrow \quad & p \wedge (q \vee r) \wedge \neg((q \wedge r) \wedge \exists yz((\neg y \wedge z) \vee (y \wedge \neg z)) \\
\leftrightarrow \quad & p \wedge (q \vee r) \wedge \neg((q \wedge r) \wedge \top) \\
\leftrightarrow \quad & p \wedge (q \vee r) \wedge \neg(q \wedge r).
\end{aligned}
$$

The sets satisfying $\mathrm{PTW}[F]$ are $\{p,q\}$ and $\{p,r\}$, which are the two stable models of F. This is an instance of a general theorem:

Pearce-Tompits-Woltran Theorem ([Pearce *et al.*, 2001], Theorem 1) *A set X of atoms occurring in F is a stable model of F iff X satisfies* $\mathrm{PTW}[F]$.

To be precise, the statement of this result in [Pearce *et al.*, 2001] refers to equilibrium models, and its reformulation above refers to stable models in the sense of Appendix A; these two concepts are equivalent to each other by Theorem 1 from [Ferraris, 2005]. A direct proof of our version of the theorem, not referring to this equivalence, is given in Section B.4 below.

Recall that the operation $F \mapsto F^*(\mathbf{v})$ replaces the atoms from $\mathbf{a}$ with the corresponding variables from $\mathbf{v}$, and that it commutes with all connectives except implication. If we drop the second conjunctive term from the clause for implication in the definition of F^* then F^* will turn into the result of substituting $\mathbf{v}$ for $\mathbf{a}$ in F, and $\mathrm{PTW}[F]$ will turn into $\mathrm{CIRC}[F;\mathbf{a}]$.

In one way, however, the operation $F \mapsto F^*(\mathbf{v})$ is essentially different from the substitution of $\mathbf{v}$ for $\mathbf{a}$: for two equivalent formulas F and G, $F^*(\mathbf{v})$ is not necessarily equivalent to $G^*(\mathbf{v})$. Here is an example:

$$
(p \rightarrow q)^* \quad = \quad (x \rightarrow y) \wedge (p \rightarrow q),
$$

$$
\begin{aligned}
(\neg p \vee q)^* \quad & = \quad (p \rightarrow \bot)^* \vee q^* \\
& = \quad ((p^* \rightarrow \bot^*) \wedge (p \rightarrow \bot)) \vee y \\
& = \quad (\neg x \wedge \neg p) \vee y \\
& \leftrightarrow \quad (x \rightarrow y) \wedge (p \rightarrow y).
\end{aligned}
$$

Applying the circumscription operator to each of two equivalent formulas gives two equivalent results; the Pearce-Tompits-Woltran transformation does not have this property.

The result of eliminating quantifiers from PTW$[F]$ (see Section B.1) can be represented using the following notation. For any subset Y of $\mathbf{a}$, by $\overrightarrow{Y}$ we denote the tuple $(Y_1, \dots, Y_n)$, where

$$Y_i = \begin{cases} \top, & \text{if } a_i \in Y; \\ \bot, & \text{otherwise.} \end{cases}$$

Then PTW$[F]$ can be written as

$$F \wedge \neg \bigvee_{Y \subseteq \mathbf{a}} ((\overrightarrow{Y} < \mathbf{a} \wedge F^*(\overrightarrow{Y}))$$

or, equivalently, as

$$F \wedge \bigwedge_{Y \subseteq \mathbf{a}} (\overrightarrow{Y} < \mathbf{a} \rightarrow \neg F^*(\overrightarrow{Y})). \tag{32}$$

B.4 Proof

In the following lemma, F is a propositional formula, and $\mathbf{a}$ is a list of distinct atoms $a_1, \dots, a_n$ containing all atoms occurring in F.

Lemma *For any subset X of $\mathbf{a}$ and any $Y \subseteq X$,*

$$Y \models F^X \textit{ iff } X \models F^*(\overrightarrow{Y}).$$

Proof by induction on F.

Case 1: F is an atom a_i, so that $F^*(\mathbf{a})$ is v_i. If $a_i \in X$ then F^X is a_i; $F^*(\overrightarrow{Y})$ is $\top$ or $\bot$ depending on whether or not $a_i \in Y$, that is, depending on whether or not Y satisfies F^X. Otherwise F^X is $\bot$; since $Y \subseteq X$, $a_i \notin Y$, so that $F^*(\overrightarrow{Y})$ is $\bot$ too.

Case 2: F is $\bot$. Each of the formulas F^X, $F^*(\overrightarrow{Y})$ is $\bot$.

Case 3: F is $G \wedge H$, so that $F^*(\overrightarrow{Y})$ is $G^*(\overrightarrow{Y}) \wedge H^*(\overrightarrow{Y})$. If X satisfies $G \wedge H$ then F^X is $G^X \wedge H^X$, and we use the induction hypothesis. Otherwise F^X is $\bot$, and X doesn't satisfy at least one of the formulas G, H. Assume, for instance, that $X \not\models G$. Then G^X is $\bot$, and, by the induction hypothesis, $X \not\models G^*(\overrightarrow{Y})$. It follows that $X \not\models F^*(\overrightarrow{Y})$.

Case 4: F is $G \lor H$, so that $F^*(\overrightarrow{Y})$ is $G^*(\overrightarrow{Y}) \lor H^*(\overrightarrow{Y})$. If X satisfies $G \lor H$ then F^X is $G^X \lor H^X$, and we use the induction hypothesis. Otherwise F^X is $\bot$, and X satisfies neither G nor H. Then each of the formulas G^X, H^X is $\bot$, and, by the induction hypothesis, X satisfies neither $G^*(\overrightarrow{Y})$ nor $H^*(\overrightarrow{Y})$. It follows that $X \not\models F^*(\overrightarrow{Y})$.

Case 5: F is $G \to H$, so that $F^*(\overrightarrow{Y})$ is

$$(G^*(\overrightarrow{Y}) \to H^*(\overrightarrow{Y})) \land (G \to H). \tag{33}$$

If X satisfies the second term $G \to H$ of (33) then F^X is $G^X \to H^X$; from the induction hypothesis we conclude that X satisfies this formula iff it satisfies the first term of (33). Otherwise F^X is $\bot$; X doesn't satisfy (33) because it doesn't satisfy the second conjunctive term.

Proof of Pearce-Tompits-Woltran Theorem. It is clear that X satisfies $\overrightarrow{Y} < \mathbf{a}$ iff Y is a proper subset of X. Using the representation (32) of PTW$[F]$, we conclude that $X \models \text{PTW}[F]$ iff

(i) $X \models F$, and

(ii) for every proper subset Y of X, $X \not\models F^*(\overrightarrow{Y})$.

It is easy to check by induction on F that $X \models F^X$ iff $X \models F$. Using this fact and the lemma above, we can restate conditions (i) and (ii) as follows:

(i') $X \models F^X$, and

(ii') for every proper subset Y of X, $Y \not\models F^X$.

This is equivalent to saying that X is a stable model of F.

References

[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.

[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, 2005] Paolo Ferraris and Vladimir Lifschitz. Mathematical foundations of answer set programming. In *We Will Show Them! Essays in Honour of Dov Gabbay*. King's College Publications, 2005. To appear.

[Ferraris, 2005] Paolo Ferraris. Answer sets for propositional theories. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 119–131, 2005.

[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, 1990] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In David Warren and Peter Szeredi, editors, *Proceedings of International Conference on Logic Programming (ICLP)*, pages 579–597, 1990.

[Lee and Lifschitz, 2003] Joohyung Lee and Vladimir Lifschitz. Loop formulas for disjunctive logic programs. In *Proceedings of International Conference on Logic Programming (ICLP)*, pages 451–465, 2003.

[Lee, 2005] Joohyung Lee. A model-theoretic counterpart of loop formulas. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 503–508, 2005.

[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.

[Lierler, 2005] Yuliya Lierler. Cmodels: SAT-based disjunctive answer set solver. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 447–452, 2005.

[Lifschitz and Razborov, 2006] Vladimir Lifschitz and Alexander Razborov. Why are there so many loop formulas? *ACM Transactions on Computational Logic*, 2006. 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.

[Lifschitz, 1994] Vladimir Lifschitz. Circumscription. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *The Handbook of Logic in AI and Logic Programming*, volume 3, pages 298–352. Oxford University Press, 1994.

[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.

[McCarthy, 1980] John McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39,171–172, 1980. Reproduced in [McCarthy, 1990].

[McCarthy, 1986] John McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 26(3):89–116, 1986. Reproduced in [McCarthy, 1990].

[McCarthy, 1990] John McCarthy. *Formalizing Common Sense: Papers by John McCarthy.* Ablex, Norwood, NJ, 1990.

[Pearce *et al.*, 2001] David Pearce, Hans Tompits, and Stefan Woltran. Encodings for equilibrium logic and logic programs with nested expressions. In *Proceedings of Portuguese Conference on Artificial Intelligence (EPIA)*, pages 306–320, 2001.

[Pearce, 1997] 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.

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

[Troelstra and Schwichtenberg, 1996] A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory.* Cambridge University Press, 1996.

[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.