A Monotonicity Theorem for Extended Logic Programs

Hudson Turner
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA
hudson@cs.utexas.edu

Abstract

Because general and extended logic programs behave nonmonotonically, it is in general difficult to predict how even minor changes to such programs will affect their meanings. This paper shows that for a restricted class of extended logic programs — those with signings — it is possible to state a fairly general theorem comparing the entailments of programs. To this end, we generalize (to the class of extended logic programs) the definition of a *signing*, first formulated by Kunen for general programs, and establish a theorem characterizing a restricted monotonicity property for signed extended programs. The theorem is formulated in terms of simple syntactic criteria on pairs of programs. To demonstrate the usefulness of this result, we use it to compare the strengths of two families of extended logic programs for commonsense reasoning about action.

1 Introduction

We would like to be able to characterize the relative strengths of logic programs in terms of simple syntactic features of the programs.

In the case of pure Prolog programs (that is, programs without negation as failure and without classical negation), this is not hard to do. For instance, by adding rules to a pure program, we can only make the program stronger. That is, given pure programs P and Q, if $P \subset Q$, then clearly program Q entails every ground atom entailed by program P.

In fact, we can easily say more about the relative entailments of pure Prolog programs. Consider the following partial order on pure program rules. Given pure rules r and r', we say that r is subsumed by r' — denoted $r \preceq r'$ — if and only if the heads of rules r and r' are identical and the body of rule r' is a subset of the body of rule r. Using this provisional definition of $\preceq$ for rules, we can define a corresponding partial order on pure programs. Given pure programs P and Q, we say that P is subsumed by Q — denoted $P \preceq Q$ — iff for every rule r in program P there is a rule r' in program Q such that $r \preceq r'$. It isn't hard to see that if $P \preceq Q$, then program Q entails every ground atom entailed by program P.

This fact is an especially simple version of the sort of monotonicity result we're after. What we see is that when we add rules to a pure program, the program becomes stronger. And also, when we subtract from the bodies of rules in a pure program, the program becomes stronger.

Unfortunately, in the presence of negation as failure, things are no longer so simple. Consider, for instance, the following pair of programs:

Program P is:	Program Q is:
$a \leftarrow \text{not } b$	$a \leftarrow \text{not } b$
$b \leftarrow \text{not } c$	$b \leftarrow \text{not } c$
$c \leftarrow d, \text{not } e$	$c \leftarrow d, \text{not } e$
$d \leftarrow \text{not } f$	$d \leftarrow \text{not } f$
$e \leftarrow$	$e \leftarrow$
	$f \leftarrow$

Program P entails exactly the atoms in $\{b, d, e\}$. Program Q entails exactly the atoms in $\{b, e, f\}$. So we add the rule $f \leftarrow$ to program P to get program Q, and yet program Q is not stronger than program P. Of course this is not surprising: we lose atom d because d "depends negatively" on f, and we don't lose atom b because b "depends positively" on f. The notion of a signing, defined by Kunen in [8] for general programs, makes this idea precise.

Let Π be a general logic program (that is, a program that includes negation as failure but not classical negation). A *signing* for Π is a set S of ground atoms such that, for any ground instance

$$A_0 \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n$$

of any rule from Π, either

$$A_0, A_1, \ldots, A_m \in S, A_{m+1}, \ldots, A_n \notin S$$

or

$$A_0, A_1, \ldots, A_m \notin S, A_{m+1}, \ldots, A_n \in S.$$ [1]

As we can see from the symmetry of this definition, it is natural to think of a signed general program as having two sides or halves, which motivates the following definition.

For a program Π with signing S, let Π_S be the set of all rules whose heads belong to S, and let $\Pi_{\overline{S}}$ be the set of all rules whose heads belong to the complement of S.

Returning to example programs P and Q above, we can see that P and Q have a common signing $S = \{a, c, d\}$; that is, S is a signing for both P

and Q. So we have

$$
\begin{aligned}
P_S = Q_S \quad &= \left[\begin{array}{lll} a & \leftarrow & \text{not } b \\ c & \leftarrow & d, \text{not } e \\ d & \leftarrow & \text{not } f \end{array}\right. \\
P_{\overline{S}} \quad &= \left[\begin{array}{lll} b & \leftarrow & \text{not } c \\ e & \leftarrow & \end{array}\right. \\
Q_{\overline{S}} \quad &= \left[\begin{array}{lll} b & \leftarrow & \text{not } c \\ e & \leftarrow & \\ f & \leftarrow & \end{array}\right.
\end{aligned}
$$

The following fact is a special case of the main result of this paper: *if signed general programs P and Q have a common signing S, and if $P_{\overline{S}} \subset Q_{\overline{S}}$ and $Q_S \subset P_S$, then P entails every ground atom in S that is entailed by Q, and Q entails every ground atom in $\overline{S}$ that is entailed by P.* We see that this theorem is applicable to example programs P and Q above.

Once we've extended in a natural way the definitions of the partial orderings $\preceq$ on rules and programs, we can state this fact a bit more generally.

As before, we intend that rule r will be subsumed by rule r' if and only if the heads of r and r' are identical and the body of r' is "a subset of" the body of rule r. More precisely, given rules r and r', we say that $r \preceq r'$ iff the following three conditions hold:

(i) the heads of rules r and r' are identical,

(ii) the set of atoms preceded by *not* in the body of rule r' is a subset of the set of atoms preceded by *not* in the body of rule r,

(iii) the set of atoms not preceded by *not* in the body of rule r' is a subset of the set of atoms not preceded by *not* in the body of rule r.

Using this definition of $\preceq$ for rules, we define as before the corresponding partial order on programs. Given programs P and Q, we say that $P \preceq Q$ iff for every rule r in program P there is a rule r' in program Q such that $r \preceq r'$.

It follows from the main theorem of this paper that *if signed general programs P and Q have a common signing S, and if $P_{\overline{S}} \preceq Q_{\overline{S}}$ and $Q_S \preceq P_S$, then P entails every ground atom in S that is entailed by Q, and Q entails every ground atom in $\overline{S}$ that is entailed by P.* Thus, given a general program Π with signing S, we know that when we add rules to $\Pi_{\overline{S}}$ or subtract rules from Π_S, program Π becomes stronger in $\overline{S}$. Furthermore, when we subtract from the bodies of rules in $\Pi_{\overline{S}}$ or add to the bodies of rules in Π_S, Π again becomes stronger in $\overline{S}$.

Our notion of entailment is based on the answer set semantics [5] (which, in the absence of classical negation, is also known as the "stable model" semantics [4]). Dung [2] has proved that, for signed general programs, answer set semantics and well-founded semantics [11] yield the same (positive) entailments.

In the next section, we define the notion of a signing for *extended* logic programs, that is, programs using both negation as failure and classical negation. This definition of a signing for extended programs extends Kunen's definition of a signing for general programs.

In fact, the particular example motivating this work involves classical negation. In the paper "Representing Actions in Extended Logic Programming", Gelfond and Lifschitz [7] introduce a simple declarative language for describing actions, called $\mathcal{A}$; and they propose a modular translation from $\mathcal{A}$ into the language of extended logic programming. Their work is presented in part as an extension of work by Apt and Bezem [1] on representing properties of actions in general logic programs. We will be able to show that the work of Gelfond and Lifschitz indeed extends the work of Apt and Bezem in the following sense: the programs of Gelfond and Lifschitz always entail at least as much about the properties of the actions described as do programs comparable to those of Apt and Bezem.

In Section 2 of this paper, we generalize the definition of a signing and state precisely the restricted monotonicity theorem for extended programs. In Section 3, we use this result to compare the strengths of two families of extended logic programs for commonsense reasoning about action. Section 4 of this paper presents a theorem concerning the relationship between the answer set semantics and the well-founded semantics in the case of signed general logic programs. This theorem facilitates the proof of the restricted monotonicity theorem, and may be of independent interest as well. The final section of this paper consists of proofs.

2 Programs and Signings

We begin with some standard definitions.

We use the symbol $\neg$ to represent classical negation. We use the term *literal* to refer to an atom possibly preceded by classical negation.

We restrict our attention to propositional programs, considering a rule with variables as an abbreviation for all of the ground instances of the rule. Therefore, from now on we will refer to ground atoms simply as atoms, and ground literals simply as literals.

An *extended* rule is a rule of the form

$$L_0 \quad \leftarrow \quad L_1, \ldots, L_m, \text{not } L_{m+1}, \ldots, \text{not } L_n \tag{1}$$

where all L_i $(i = 0, \ldots, n)$ are literals. An *extended* program is a set of extended rules.

There are two important special cases:

A *positive* rule is a rule of the form

$$L_0 \leftarrow L_1, \ldots, L_m$$

where all L_i $(i = 0, \ldots, m)$ are literals. A *positive* program is a set of positive rules.

A *general* rule is a rule of the form

$$A_0 \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n$$

where all A_i $(i = 0, \ldots, n)$ are atoms. A *general* program is a set of general rules.

Let Π be an extended program. By atoms(Π) we denote the set of atoms that occur in Π (in either negated or non-negated form). By literals(Π) we denote the set of literals occuring in Π.

Given a rule r as in (1), we define the following: $\text{head}(r) = \{L_0\}$, $\text{pos}(r) = \{L_1, \ldots, L_m\}$, and $\text{neg}(r) = \{L_{m+1}, \ldots, L_n\}$. We will refer to the expressions not $L_{m+1}, \ldots,$ not L_n as the *negative subgoals* of rule r.

A set B of literals is *closed under* the rules of positive program Π if $\forall r \in \Pi \,.\, \text{pos}(r) \subset B \Rightarrow \text{head}(r) \cap B \neq \emptyset$.

A set B of literals is *logically closed* with respect to a program Π if either B contains no complementary literals or $B = \text{literals}(\Pi)$.

For a positive program Π, a set B of literals is an *answer set* for Π iff B is a minimal set of literals closed under Π and logically closed w.r.t Π. We denote by $\alpha\Pi$ the (unique) answer set for positive program Π.

For an extended program Π and a set of literals X, the reduct operator $X \mapsto \Pi^X$ transforms program Π into the positive program Π^X obtained from Π by first deleting every rule $r \in \Pi$ such that $\text{neg}(r) \cap X \neq \emptyset$, and then, for each remaining rule $r \in \Pi$, deleting all the negative subgoals of r.

A set B of literals is an *answer set* for an extended program Π iff $B = \alpha\Pi^B$. An extended program Π *entails* exactly those literals in literals(Π) that are included in every answer set for Π.

Now we introduce a few new definitions, including the definition of a signing for extended programs. Then we will be able to state the main theorem of this paper.

Definition. Given rules r and r',
$r \preceq r'$ iff $\text{head}(r') = \text{head}(r)$ and $\text{pos}(r') \subset \text{pos}(r)$ and $\text{neg}(r') \subset \text{neg}(r)$.

Definition. Given logic programs P and Q,
$P \preceq Q$ iff for each rule $r \in P$ there is a rule $r' \in Q$ such that $r \preceq r'$.

Definition. A set S of atoms is a *signing* for an extended logic program Π if no atom in S appears negated in literals(Π) and

$$\forall r \in \Pi \,.\, \Big(\text{head}(r) \cup \text{pos}(r) \subset S \wedge \text{neg}(r) \subset \overline{S}\Big) \\ \vee \Big(\text{head}(r) \cup \text{pos}(r) \subset \overline{S} \wedge \text{neg}(r) \subset S\Big) \ ,$$

where $\overline{S}$ denotes the complement of S with respect to the set of all literals.

The following extended program P has a signing $S = \{b\}$:

$$\begin{array}{rcl} a & \leftarrow & \text{not } b, \\ b & \leftarrow & \text{not } a, \\ \neg a & \leftarrow & . \end{array}$$

Observe that $\{a, \neg a\}$ is *not* a signing for program P, since $\neg a$ is not an atom. Neither is $\{a\}$ a signing for P, since $\neg a \in \text{literals}(P)$. And in general, the definition of a signing for extended programs is asymmetric in the following sense: if a set S of atoms is a signing for a program Π that includes classical negation, the set of atoms in $\overline{S}$ will not be a signing for Π, because some atom in $\overline{S}$ will appear negated in literals(Π). For general programs though, the definition is symmetric; that is, if S is a signing for a general program Π, then the set of atoms in $\overline{S}$ will also be a signing for Π. For general programs, this definition is equivalent to Kunen's. Before we demonstrate why this asymmetry is useful, we state formally the restricted monotonicity theorem for extended programs:

Theorem 1 *For extended logic programs P and Q with common signing S, if $P_{\overline{S}} \preceq Q_{\overline{S}}$ and $Q_S \preceq P_S$ and literals$(P) \cap \overline{S} \subset$ literals$(Q) \cap \overline{S}$, then program Q entails every literal in $\overline{S}$ that is entailed by program P.*

Now, of course, when we wish to generalize the concept of a signing to the class of extended programs, the first possibility we consider is simply to employ Kunen's definition, but applied now to literals instead of atoms. Unfortunately, under this straightforward, symmetric generalization of the definition, the restricted monotonicity property for signed programs does not hold. The following pair of extended programs illustrates this shortcoming:

Program P is:

$$\begin{array}{rcl} a & \leftarrow & \text{not } \neg b, \\ \neg a & \leftarrow & \text{not } b, \\ b & \leftarrow & \text{not } a, \text{not } c, \\ \neg b & \leftarrow & \text{not } a, \\ c & \leftarrow & \text{not } b. \end{array}$$

Program Q is:

$$\begin{array}{rcl} a & \leftarrow & \text{not } \neg b, \\ \neg a & \leftarrow & \text{not } b, \\ b & \leftarrow & \text{not } a, \text{not } c, \\ \neg b & \leftarrow & \text{not } a, \\ c & \leftarrow & \text{not } b, \\ b & \leftarrow & . \end{array}$$

Program P has a single answer set, $\{\neg a, \neg b, c\}$. Program Q has a single answer set, $\{a, b\}$. Clearly we'd be hard-pressed to identify monotonicity here. Yet under the hypothetical, symmetric definition of a signing for extended programs, the set $S = \{a, \neg a, c\}$ would be a common signing for programs P and Q, in which case the monotonicity theorem would be falsified.

3 Restricted Monotonicity in Logic Programs for Commonsense Reasoning About Action

In the paper "Representing Actions in Extended Logic Programming", Gelfond and Lifschitz [7] introduce a simple declarative language for describing actions, called $\mathcal{A}$; and they propose a modular translation from $\mathcal{A}$ into the language of extended logic programming. Their work is presented in part as an extension of work by Apt and Bezem [1] (among others) on representing properties of actions in general logic programs.

Given Theorem 1, we can easily show that the work of Gelfond and Lifschitz indeed extends the work of Apt and Bezem in the following sense: the programs of Gelfond and Lifschitz always entail at least as much about the properties of the actions described as do programs comparable to those of Apt and Bezem.

In the language $\mathcal{A}$, a description of an action domain is a set D of propositions of two kinds: value-propositions — which specify the value of a fluent in a particular situation; and effect-propositions — which describe the effect of an action on a fluent.

We will briefly describe the syntax of $\mathcal{A}$. As for the semantics of $\mathcal{A}$, we simply remark that they are based in a straightforward manner on deterministic finite automata, and that the results of this semantics are generally intuitive. (See [7] for the full story.)

Begin with two disjoint non-empty sets of symbols, called *fluent names* and *action names*. A *fluent expression* is a fluent name possibly preceded by $\neg$. A *value-proposition* is an expression of the form F **after** $A_1; \ldots; A_m$, where F is a fluent expression, and $A_1, \ldots, A_m$ $(m \geq 0)$ are action names. If $m = 0$, we write instead **initially** F. An *effect-proposition* is an expression of the form A **causes** F **if** $P_1, \ldots, P_n$, where A is an action name, and each of $F, P_1, \ldots, P_n$ $(n \geq 0)$ is a fluent expression. About this proposition we say that it describes the *effect* of A on F, and that $P_1, \ldots, P_n$ are its *preconditions*. If $n = 0$, we drop **if** and write simply A **causes** F.

Now, if we're interested only in temporal projection problems, in which the given value-propositions refer only to the initial situation, then we can specify a very simple translation from domain descriptions D in the language of $\mathcal{A}$ into extended logic programs $\pi_{\text{forward}}D$. And although of course the work of Apt and Bezem was not presented as a translation from the language $\mathcal{A}$ and did not use classical negation, the programs $\pi_{\text{forward}}D$ correspond nicely to the general logic programs that Apt and Bezem proposed.

Thus, for each domain D there is an extended program $\pi_{\text{forward}}D$ which consists of a single rule for each value-proposition in D, a pair of rules for each effect-proposition in D, and a standard pair of rules expressing inertial properties. In specifying this translation, we rely on the following convention: for any fluent name G and situation term t, $\mathit{Holds}(\neg G, t)$ stands for $\neg \mathit{Holds}(G, t)$.

A value-proposition of the form **initially** F is translated into the rule

$$\mathit{Holds}(F, S_0) \leftarrow \ . \tag{2}$$

A value-proposition of the form F **after** $A_1; \ldots; A_m$ is translated into the rule

$$\mathit{Holds}(F, \mathit{Result}(A_m, \mathit{Result}(A_{m-1}, \ldots, \mathit{Result}(A_1, S_0) \ldots))) \leftarrow \ . \tag{3}$$

An effect-proposition of the form A **causes** F **if** $P_1, \ldots, P_n$ is translated into two rules. The first of them is

$$\mathit{Holds}(F, \mathit{Result}(A, s)) \leftarrow \mathit{Holds}(P_1, s), \ldots, \mathit{Holds}(P_n, s). \tag{4}$$

The second rule is

$$\mathit{Noninertial}(|F|, A, s) \leftarrow \text{not } \overline{\mathit{Holds}(P_1, s)}, \ldots, \text{not } \overline{\mathit{Holds}(P_n, s)}, \tag{5}$$

where $\overline{\mathit{Holds}(P_i, s)}$ is the literal complementary to $\mathit{Holds}(P_i, s)$, and where $|F|$ is the fluent name corresponding to the fluent expression F.

Finally, the translation $\pi_{\text{forward}}D$ includes two inertial rules:

$$\begin{array}{rcl} \mathit{Holds}(f, \mathit{Result}(a, s)) & \leftarrow & \mathit{Holds}(f, s), \text{not } \mathit{Noninertial}(f, a, s). \\ \neg\mathit{Holds}(f, \mathit{Result}(a, s)) & \leftarrow & \neg\mathit{Holds}(f, s), \text{not } \mathit{Noninertial}(f, a, s). \end{array} \tag{6}$$

Observe that all of the rules in $\pi_{\text{forward}}D$ are apparently intended for reasoning forward in time.

Suppose instead that we wish to reason about domains in which the given value-propositions may refer to non-initial situations. Faced with such domains, we may wish to formulate additional rules intended for reasoning backward in time. To this end, Gelfond and Lifschitz proposed their translation from domain descriptions D into extended programs πD. Each program πD is a superset of the corresponding program $\pi_{\text{forward}}D$. So, in addition to the rules in $\pi_{\text{forward}}D$, we include the following backward reasoning rules in program πD:

For an effect-proposition of the form A **causes** F **if** $P_1, \ldots, P_n$, we add for each i, $1 \leq i \leq n$, the credit assignment rule

$$\mathit{Holds}(P_i, s) \leftarrow \overline{\mathit{Holds}(F, s)}, \mathit{Holds}(F, \mathit{Result}(A, s)).\ ^{2} \tag{7}$$

and the blame assignment rule

$$\begin{array}{rl} \overline{\mathit{Holds}(P_i, s)} \leftarrow & \overline{\mathit{Holds}(F, \mathit{Result}(A, s))}, \mathit{Holds}(P_1, s), \ldots, \mathit{Holds}(P_{i-1}, s), \\ & \mathit{Holds}(P_{i+1}, s), \ldots, \mathit{Holds}(P_n, s). \end{array} \tag{8}$$

We also include in πD another pair of inertial rules:

$$\begin{array}{rcl} \mathit{Holds}(f, s) & \leftarrow & \mathit{Holds}(f, \mathit{Result}(a, s)), \text{not } \mathit{Noninertial}(f, a, s). \\ \neg\mathit{Holds}(f, s) & \leftarrow & \neg\mathit{Holds}(f, \mathit{Result}(a, s)), \text{not } \mathit{Noninertial}(f, a, s). \end{array} \tag{9}$$

On the one hand, it may seem intuitively clear that, for any domain description D in the language of $\mathcal{A}$, program πD should entail at least as many *Holds* and $\neg$*Holds* literals as does program $\pi_{\text{forward}}D$. On the other hand, such an assertion needs a proof. And in fact, it is likely that this intuition is based on experience with monotonic formalisms, which may be misleading here.

For us, the question about relative entailments of these two translations from the language of $\mathcal{A}$ arose particularly in response to the following simple domain description D:

$$\begin{array}{l} \textbf{initially } \neg F \\ A \textbf{ causes } F \text{ if } G \end{array}$$

Even though this is a temporal projection problem, in which the given value-propositions refer only to the initial situation, it nevertheless happens that the additional rules in πD for reasoning backward in time still affect the meaning of the program. Thus, program πD has two answer sets, while program $\pi_{\text{forward}} D$ has just one answer set.

The program $\pi_{\text{forward}} D$ is:

$$\begin{array}{rcl} \neg \mathit{Holds}(F, S_0) & \leftarrow & . \\ \mathit{Holds}(F, \mathit{Result}(A, s)) & \leftarrow & \mathit{Holds}(G, s). \\ \mathit{Noninertial}(F, A, s) & \leftarrow & \mathit{not}\ \neg \mathit{Holds}(G, s). \\ \mathit{Holds}(f, \mathit{Result}(a, s)) & \leftarrow & \mathit{Holds}(f, s), \text{not } \mathit{Noninertial}(f, a, s). \\ \neg \mathit{Holds}(f, \mathit{Result}(a, s)) & \leftarrow & \neg \mathit{Holds}(f, s), \text{not } \mathit{Noninertial}(f, a, s). \end{array}$$

Program $\pi_{\text{forward}} D$ has a single answer set consisting of $\neg \mathit{Holds}(F, S_0)$ and all ground instances of $\mathit{Noninertial}(F, A, s)$.[3]

By comparison, program πD is:

$$\begin{array}{rcl} \neg \mathit{Holds}(F, S_0) & \leftarrow & . \\ \mathit{Holds}(F, \mathit{Result}(A, s)) & \leftarrow & \mathit{Holds}(G, s). \\ \mathit{Noninertial}(F, A, s) & \leftarrow & \text{not } \neg \mathit{Holds}(G, s). \\ \mathit{Holds}(G, s) & \leftarrow & \neg \mathit{Holds}(F, s),\ \mathit{Holds}(F, \mathit{Result}(A, s)). \\ \neg \mathit{Holds}(G, s) & \leftarrow & \neg \mathit{Holds}(F, \mathit{Result}(A, s)). \\ \mathit{Holds}(f, \mathit{Result}(a, s)) & \leftarrow & \mathit{Holds}(f, s), \text{not } \mathit{Noninertial}(f, a, s). \\ \neg \mathit{Holds}(f, \mathit{Result}(a, s)) & \leftarrow & \neg \mathit{Holds}(f, s), \text{not } \mathit{Noninertial}(f, a, s). \\ \mathit{Holds}(f, s) & \leftarrow & \mathit{Holds}(f, \mathit{Result}(a, s)), \text{not } \mathit{Noninertial}(f, a, s). \\ \neg \mathit{Holds}(f, s) & \leftarrow & \neg \mathit{Holds}(f, \mathit{Result}(a, s)), \text{not } \mathit{Noninertial}(f, a, s). \end{array}$$

Program πD has two answer sets. One consists of $\neg \mathit{Holds}(F, S_0)$ and all ground instances of $\mathit{Noninertial}(F, A, s)$. The other consists of all ground instances of $\neg \mathit{Holds}(F, s)$ and $\neg \mathit{Holds}(G, s)$.[4]

Despite the unanticipated result in this specific case, we have the following satisfactory general result comparing the entailments of the programs πD and $\pi_{\text{forward}} D$ for arbitrary domains D:

Proposition 1 *Given any domain description D in the language of $\mathcal{A}$, the extended logic program πD entails at least every* Holds *ground literal and every* $\neg$Holds *ground literal entailed by the extended logic program* $\pi_{\text{forward}} D$.

Proof.
Let D be a domain description in the language of $\mathcal{A}$.

Let $S = \{b : b \text{ is a } \textit{Noninertial} \text{ atom in literals}(\pi D)\}$.

Clearly S is a common signing for extended programs $\pi_{\text{forward}}D$ and πD.

It is also clear that $(\pi_{\text{forward}}D\ )_{\overline{S}} \subset (\pi D)_{\overline{S}}$ and that $(\pi D)_S = (\pi_{\text{forward}}D\ )_S$. Therefore, we have $(\pi_{\text{forward}}D\ )_{\overline{S}} \preceq (\pi D)_{\overline{S}}$ and $(\pi D)_S \preceq (\pi_{\text{forward}}D\ )_S$.

Finally, it is clear that $\text{literals}(\pi_{\text{forward}}D\ ) \cap \overline{S} \subset \text{literals}(\pi D) \cap \overline{S}$.

Thus, by Theorem 1, πD entails at least every ground literal in $\overline{S}$ that is entailed by $\pi_{\text{forward}}D$.
□

4 Answer Set and Well-Founded Semantics

We exploit in this paper the close relationship between the answer set semantics for general logic programs [4] and the well-founded semantics for general logic programs [11]. This section restates in a form convenient for our purposes a number of previously-known results in the declarative semantics of general logic programs. At the close of this section, we state a new result for signed general programs.

In this section, Π stands for any general logic program. Recall that a set B of ground atoms is an answer set for Π iff $B = \alpha\Pi^B$.

Let $\Gamma X = \alpha\Pi^X$. Observe that the answer sets of Π can be characterized as the fixpoints of Γ. It is easy to see that Γ is anti-monotone. Consequently, Γ^2 is monotone. Because Γ^2 is monotone, we know by the Knaster–Tarski theorem [10] that Γ^2 has a least fixpoint, $\text{lfp}(\Gamma^2)$, and a greatest fixpoint, $\text{gfp}(\Gamma^2)$.

Let $\text{WF}_\perp(\Pi)$ denote $\text{lfp}(\Gamma^2)$; and let $\text{WF}_\top(\Pi)$ denote $\text{gfp}(\Gamma^2)$. These two sets — $\text{WF}_\perp(\Pi)$ and $\text{WF}_\top(\Pi)$ — capture essential information about the well-founded semantics of a general program Π. That is, under the well-founded semantics, when a ground atom b is submitted as a query: the answer is "yes" when $b \in \text{WF}_\perp(\Pi)$; "unknown" when $b \in \text{WF}_\top(\Pi)\backslash\text{WF}_\perp(\Pi)$; and "no" when $b \notin \text{WF}_\top(\Pi)$.

For all answer sets X for Π,

$$\text{WF}_\perp(\Pi) \subset X \subset \text{WF}_\top(\Pi),$$

because each fixpoint of Γ is a fixpoint of Γ^2. This fact indicates that the well-founded semantics can be seen as an approximation to the answer set semantics, identifying lower and upper bounds on the answer sets for a general program. As we will see in Theorem 2 and the remainder of this paper, the relationship between the answer set semantics and the well-founded semantics grows even closer in the case of signed programs.

Theorem 2 *For a general program* Π *with signing* S*, the following are among the answer sets for* Π*:*

$$\begin{array}{ll} (i) & WF_{\perp}(\Pi) \cup (WF_{\top}(\Pi) \cap \overline{S}) \\ (ii) & WF_{\perp}(\Pi) \cup (WF_{\top}(\Pi) \cap S)\ . \end{array}$$

Theorem 2 is of particular interest because it gives a direct characterization, in terms of the well-founded semantics, of two (possibly identical) answer sets for any signed general program. As we will see, this characterization facilitates the proof of Theorem 1. In fact, these are the two most important answer sets of a signed general program Π, in the sense that Π entails exactly their intersection. So we see, as has been established previously in [2], that a signed general program Π entails exactly the ground atoms included in $\mathrm{WF}_{\perp}(\Pi)$.[5]

Theorem 2 also provides an unusually direct proof of another previously established result: a signed general logic program has at least one answer set. A closely related proof of the existence of answer sets for signed general programs is presented in [7]. There the authors define a monotone operator ϕ associated with each signed general program Π and show that every fixpoint of ϕ corresponds to an answer set for Π. In particular, it turns out that $\mathrm{lfp}(\phi)$ corresponds to $\mathrm{WF}_{\perp}(\Pi) \cup (\mathrm{WF}_{\top}(\Pi) \cap S)$, and $\mathrm{gfp}(\phi)$ corresponds to $\mathrm{WF}_{\perp}(\Pi) \cup (\mathrm{WF}_{\top}(\Pi) \cap \overline{S})$. In [3], Fages establishes the existence of answer sets for a much larger class of general logic programs. Fages shows that *order-consistent* general programs have answer sets.[6]

Observe that a signed general program may have additional answer sets that do not correspond, in the manner of Theorem 2, to any signing for the program. For example, $\{a, d\}$ is among the answer sets for the following signed program:

$$\begin{array}{lll} a & \leftarrow & \text{not } b, \\ b & \leftarrow & \text{not } a, \\ c & \leftarrow & \text{not } b, \text{not } d, \\ d & \leftarrow & \text{not } c. \end{array}$$

5 Proofs

We begin with the proof of Theorem 2, which uses the following lemma:

Lemma 1 *For a general program* Π *with signing* S*,*

$$\begin{array}{ll} (i) & \alpha\Pi^X = \alpha\Pi_S^{X\cap\overline{S}} \cup \alpha\Pi_{\overline{S}}^{X\cap S} \\ (ii) & \alpha\Pi^X \cap S = \alpha\Pi_S^{X\cap\overline{S}} \\ (iii) & \alpha\Pi^X \cap \overline{S} = \alpha\Pi_{\overline{S}}^{X\cap S}\ . \end{array}$$

Proof. Observe that $\Pi = \Pi_S \cup \Pi_{\overline{S}}$. Thus, by the definition of the reduct operator, we have $\Pi^X = (\Pi_S \cup \Pi_{\overline{S}})^X = \Pi_S^X \cup \Pi_{\overline{S}}^X$.

By the definition of a signing, we know that $\forall r \in \Pi_S \;.\; \mathrm{neg}(r) \subset \overline{S}$. So $\Pi_S^X = \Pi_S^{X \cap \overline{S}}$. Likewise, $\Pi_{\overline{S}}^X = \Pi_{\overline{S}}^{X \cap S}$. Also by the definition of a signing we have $\forall r \in \Pi_S \;.\; \mathrm{head}(r) \cup \mathrm{pos}(r) \subset S$. So again by the definition of the reduct operator, $\mathrm{atoms}(\Pi_S^{X \cap \overline{S}}) \subset S$, and clearly, $\alpha\Pi_S^{X \cap \overline{S}} \subset S$. Likewise, $\mathrm{atoms}(\Pi_{\overline{S}}^{X \cap S}) \subset \overline{S}$, and $\alpha\Pi_{\overline{S}}^{X \cap S} \subset \overline{S}$.

Given these observations, it is easy to verify that $\alpha\Pi^X = \alpha\left(\Pi_S^{X \cap \overline{S}} \cup \Pi_{\overline{S}}^{X \cap S}\right) = \alpha\Pi_S^{X \cap \overline{S}} \cup \alpha\Pi_{\overline{S}}^{X \cap S}$, with $\alpha\Pi^X \cap S = \alpha\Pi_S^{X \cap \overline{S}}$ and $\alpha\Pi^X \cap \overline{S} = \alpha\Pi_{\overline{S}}^{X \cap S}$.
□

Theorem 2 *For a general program* Π *with signing* S*, the following are among the answer sets for* Π*:*

$$\begin{array}{ll} \textit{(i)} & WF_{\perp}(\Pi) \cup (WF_{\top}(\Pi) \cap \overline{S}) \\ \textit{(ii)} & WF_{\perp}(\Pi) \cup (WF_{\top}(\Pi) \cap S)\;. \end{array}$$

Proof. We give a proof for (i); (ii) follows by symmetry, because if S is a signing for a general logic program Π, then so is $\overline{S}$.

Let $A = \mathrm{WF}_{\perp}(\Pi) \cup \left(\mathrm{WF}_{\top}(\Pi) \cap \overline{S}\right)$. Observe that $A \cap S = \mathrm{WF}_{\perp}(\Pi) \cap S$ and $A \cap \overline{S} = \mathrm{WF}_{\top}(\Pi) \cap \overline{S}$. We will show that $A = \Gamma A$, which is to say that A is an answer set for Π.

Now, because $\mathrm{WF}_{\top}(\Pi)$ a fixpoint of Γ^2, we know that $\Gamma\left(\mathrm{WF}_{\top}(\Pi)\right)$ is also a fixpoint of Γ^2. Therefore, by the anti-monotonicity of Γ, along with the fact that $\mathrm{WF}_{\perp}(\Pi)$ and $\mathrm{WF}_{\top}(\Pi)$ are the least and greatest fixpoints of Γ^2, we can conclude that $\mathrm{WF}_{\perp}(\Pi) = \Gamma\left(\mathrm{WF}_{\top}(\Pi)\right)$.

By the definition of Γ, $\Gamma\left(\mathrm{WF}_{\top}(\Pi)\right) = \alpha\Pi^{\mathrm{WF}_{\top}(\Pi)}$.

Thus, by Lemma 1(ii), we have $\Gamma(\mathrm{WF}_{\top}(\Pi)) \cap S = \alpha\Pi_S^{\mathrm{WF}_{\top}(\Pi) \cap \overline{S}}$.

Combining these observations,

$$\begin{aligned} A \cap S &= \mathrm{WF}_{\perp}(\Pi) \cap S \\ &= \Gamma\left(\mathrm{WF}_{\top}(\Pi)\right) \cap S \\ &= \alpha\Pi_S^{\mathrm{WF}_{\top}(\Pi) \cap \overline{S}} \\ &= \alpha\Pi_S^{A \cap \overline{S}}. \end{aligned}$$

By symmetric reasoning, we have $A \cap \overline{S} = \alpha\Pi_{\overline{S}}^{A \cap S}$. Finally, by the definition of Γ and Lemma 1(i), along with the foregoing observations,

$$\begin{aligned} \Gamma A &= \alpha\Pi_S^{A \cap \overline{S}} \cup \alpha\Pi_{\overline{S}}^{A \cap S} \\ &= (A \cap S) \cup (A \cap \overline{S}) \\ &= A. \end{aligned}$$

□

Now we begin the proof of Theorem 1, which relies on the following restricted monotonicity theorem for signed *general* programs:

Theorem 3 *For general programs P and Q with common signing S, if $P_S \preceq Q_S$ and $Q_{\overline{S}} \preceq P_{\overline{S}}$, then $WF_\bot(P) \cap S \subset WF_\bot(Q) \cap S$.*

Because the set of atoms in $\overline{S}$ will also be a common signing for general programs P and Q, we have:

Corollary 1 *For general programs P and Q with common signing S, if $P_{\overline{S}} \preceq Q_{\overline{S}}$ and $Q_S \preceq P_S$, then $WF_\bot(P) \cap \overline{S} \subset WF_\bot(Q) \cap \overline{S}$.*

Our proof of Theorem 3 requires a number of lemmas.

Lemma 2 *Let P and Q be general programs. Let X and Y be sets of atoms. If $P \preceq Q$ and $Y \subset X$, then $P^X \preceq Q^Y$.*

Proof.
By the definition of $\preceq$ for programs, we have $\forall r \in P \,.\, \exists r' \in Q \,.\, r \preceq r'$.
Applying the definition of $\preceq$ for rules, this gives us
$\forall r \in P \,.\, \exists r' \in Q \,.\, \mathrm{head}(r') = \mathrm{head}(r) \wedge \mathrm{pos}(r') \subset \mathrm{pos}(r) \wedge \mathrm{neg}(r') \subset \mathrm{neg}(r)$.
In particular, given the set of atoms X, with $P \preceq Q$, what's important in comparing the reducts P^X and Q^X is the following observation:

$$\forall r \in P \,.\, \mathrm{neg}(r) \cap X = \emptyset \Rightarrow \exists r' \in Q \,.\, \mathrm{neg}(r') \cap X = \emptyset \;\wedge \mathrm{head}(r') = \mathrm{head}(r) \\ \wedge \mathrm{pos}(r') \subset \mathrm{pos}(r)\ .$$

Therefore, we conclude that for any set of atoms X, $P \preceq Q \Rightarrow P^X \preceq Q^X$. And since $Y \subset X$, we know by the anti-monotonicity of the reduct operator that $Q^X \subset Q^Y$, which implies that $Q^X \preceq Q^Y$. Thus, because $\preceq$ is clearly transitive, we have $P^X \preceq Q^Y$.
□

Lemma 3 *Let P and Q be positive general programs. If $P \preceq Q$, then $\alpha P \subset \alpha Q$.*

Proof. Obvious from the definitions, since positive general programs behave monotonically. □

Now we introduce a useful pair of monotone operators:

Let Π be a general program with signing S.
Let $\Delta^\Pi_S X = \alpha\Pi_S^{\alpha\Pi^X_{\overline{S}}}$, and let $\Delta^\Pi_{\overline{S}} X = \alpha\Pi_{\overline{S}}^{\alpha\Pi^X_S}$.

Lemma 4 *The operators Δ^Π_S and $\Delta^\Pi_{\overline{S}}$ are monotone.*

Proof. By the monotonicity of α and the anti-monotonicity of the reduct operator. □

Lemma 5 *X is a fixpoint of Γ^2 iff $X \cap S$ is a fixpoint of Δ_S^Π and $X \cap \overline{S}$ is a fixpoint of $\Delta_{\overline{S}}^\Pi$.*

Proof. By the definition of Γ, we have $\Gamma X = \alpha\Pi^X$.

By Lemma 1(ii), $\Gamma X \cap S = \alpha\Pi_S^{X\cap\overline{S}}$, and by Lemma 1(iii), $\Gamma X \cap \overline{S} = \alpha\Pi_{\overline{S}}^{X\cap S}$.

Again by the definition of Γ, $\Gamma^2 X = \alpha\Pi^{\Gamma X}$. And again by Lemma 1(ii), $\Gamma^2 X \cap S = \alpha\Pi_S^{\Gamma X\cap\overline{S}}$, and by Lemma 1(iii), $\Gamma^2 X \cap \overline{S} = \alpha\Pi_{\overline{S}}^{\Gamma X\cap S}$.

So to sum up, $\Gamma^2 X \cap S = \alpha\Pi_S^{\Gamma X\cap\overline{S}} = \alpha\Pi_S^{\alpha\Pi_{\overline{S}}^{X\cap S}} = \Delta_S^\Pi(X \cap S)$.

Likewise, $\Gamma^2 X \cap \overline{S} = \alpha\Pi_{\overline{S}}^{\Gamma X\cap S} = \alpha\Pi_{\overline{S}}^{\alpha\Pi_S^{X\cap\overline{S}}} = \Delta_{\overline{S}}^\Pi(X \cap \overline{S})$.

Therefore we can conclude

$$\begin{aligned} \Gamma^2 X = X \quad &\text{iff} \quad \Gamma^2 X \cap S = X \cap S \text{ and } \Gamma^2 X \cap \overline{S} = X \cap \overline{S} \\ &\text{iff} \quad \Delta_S^\Pi(X \cap S) = X \cap S \text{ and } \Delta_{\overline{S}}^\Pi(X \cap \overline{S}) = X \cap \overline{S}. \end{aligned}$$

□

Lemma 6

$$\mathit{lfp}(\Delta_S^\Pi) = \mathit{WF}_\perp(\Pi) \cap S$$

Proof. By Lemma 4, the operators Δ_S^Π and $\Delta_{\overline{S}}^\Pi$ are monotone, so by the Knaster-Tarski theorem each has a least and greatest fixpoint. Since $\mathrm{WF}_\perp(\Pi)$ is a fixpoint of Γ^2, it follows by Lemma 5 that $\mathrm{WF}_\perp(\Pi) \cap S$ is a fixpoint of Δ_S^Π. It remains to show that for any fixpoint X of Δ_S^Π, $\mathrm{WF}_\perp(\Pi) \cap S \subset X$. So let $X \subset S$ be a fixpoint of Δ_S^Π, and let $Y \subset \overline{S}$ be a fixpoint of $\Delta_{\overline{S}}^\Pi$. By Lemma 5, $X \cup Y$ is a fixpoint of Γ^2. But by definition, $\mathrm{WF}_\perp(\Pi)$ is the least fixpoint of Γ^2. So $\mathrm{WF}_\perp(\Pi) \subset X \cup Y$, and therefore $\mathrm{WF}_\perp(\Pi) \cap S \subset (X \cup Y) \cap S = X$. □

Lemma 7 *Let P and Q be general programs with common signing S.*
Let X be a set of atoms. Let $\Delta_S^P X = \alpha P_S^{\alpha P_{\overline{S}}^X}$; let $\Delta_S^Q X = \alpha Q_S^{\alpha Q_{\overline{S}}^X}$.
If $P_S \preceq Q_S$ and $Q_{\overline{S}} \preceq P_{\overline{S}}$, then $\Delta_S^P X \subset \Delta_S^Q X$.

Proof. By Lemma 2, since $Q_{\overline{S}} \preceq P_{\overline{S}}$, we know that $Q_{\overline{S}}^X \preceq P_{\overline{S}}^X$. So by Lemma 3 we have $\alpha Q_{\overline{S}}^X \subset \alpha P_{\overline{S}}^X$. Again by Lemma 2, since $P_S \preceq Q_S$ and $\alpha Q_{\overline{S}}^X \subset \alpha P_{\overline{S}}^X$, we can conclude that $P_S^{\alpha P_{\overline{S}}^X} \preceq Q_S^{\alpha Q_{\overline{S}}^X}$. And once more by Lemma 3, we have $\alpha P_S^{\alpha P_{\overline{S}}^X} \subset \alpha Q_S^{\alpha Q_{\overline{S}}^X}$; that is to say, $\Delta_S^P X \subset \Delta_S^Q X$.
□

Now we're ready to prove Theorem 3.

Proof. (of Theorem 3)
Let P and Q be general programs with joint signing S, such that $P_S \preceq Q_S$ and $Q_{\overline{S}} \preceq P_{\overline{S}}$. We wish to show that $\mathrm{WF}_\perp(P) \cap S \subset \mathrm{WF}_\perp(Q) \cap S$.

Let $\Delta^P_S X = \alpha P_S^{\alpha P\frac{X}{S}}$; and let $\Delta^Q_S X = \alpha Q_S^{\alpha Q\frac{X}{S}}$. By Lemma 4, Δ^P_S and Δ^Q_S are monotone.

By Lemma 6, $\mathrm{lfp}(\Delta^P_S) = \mathrm{WF}_\perp(P) \cap S$ and $\mathrm{lfp}(\Delta^Q_S) = \mathrm{WF}_\perp(Q) \cap S$.

By the Knaster-Tarski theorem, the least fixpoint of a monotone operator is also the least pre-fixpoint of that operator, which coincides with the intersection of all pre-fixpoints of the operator. Therefore,

$$\mathrm{WF}_\perp(P) \cap S = \bigcap_{\Delta^P_S X \subset X} X \qquad \text{and} \qquad \mathrm{WF}_\perp(Q) \cap S = \bigcap_{\Delta^Q_S X \subset X} X \,.$$

Thus, in order to demonstrate that $\mathrm{WF}_\perp(P) \cap S \subset \mathrm{WF}_\perp(Q) \cap S$, it is sufficient to demonstrate that each pre-fixpoint of Δ^Q_S is also a pre-fixpoint of Δ^P_S. That is, we must show that $\Delta^Q_S X \subset X \Rightarrow \Delta^P_S X \subset X$. So, assume that $\Delta^Q_S X \subset X$. By Lemma 7 we can conclude that $\Delta^P_S X \subset \Delta^Q_S X$. It follows that $\Delta^P_S X \subset X$.
$\Box$

Before we go on to prove Theorem 1, we must introduce a few more definitions and observations relating extended logic programs to their general counterparts.

We say that a set B of literals is *consistent* if B doesn't contain a complementary pair of literals. An extended logic program Π is *consistent* if Π has an answer set B that is consistent.

Given a set B of literals, we define a corresponding set of atoms
$B^+ = \{b : b \text{ is a non-negated atom in } B\} \cup \{b' : b \text{ is a negated atom in } B\}$.
The intention here is that the b' atoms will be new to the language. So we use the prime symbol to create new atoms. Given a set B of atoms, we define a corresponding set of literals $B^- = \{b : b \in B\} \cup \{\neg b : b' \in B\}$.

We say that a set of atoms B^+ contains a *bad pair* if there is a $b \in B$ such that both b,$b' \in B^+$. Note that B^+ contains a bad pair if and only if B is inconsistent.

Given an extended logic program Π, we define the corresponding general program Π^+ as the program that results from replacing each literal in Π with its corresponding atom in $(\mathrm{literals}(\Pi))^+$.

For an extended program Π, let $\mathrm{LB}(\Pi)$ denote $(\mathrm{WF}_\perp(\Pi^+))^-$; and let $\mathrm{UB}(\Pi)$ denote $(\mathrm{WF}_\top(\Pi^+))^-$.

Finally, note that if S is a signing for extended program Π, then S is also a signing for Π^+.

Proposition 2 *Let Π be an extended program.*

(i) Program Π is consistent iff Π^+ has an answer set that contains no bad pairs.

(ii) *If* Π *is consistent, then* B *is an answer set for* Π *iff* B^+ *is an answer set for* Π^+ *and* B^+ *contains no bad pairs.*

(iii) *If* Π *is not consistent, then literals*(Π) *is the only* possible *answer set for* Π.

Proof. Immediate by the definitions and Proposition 2 of [6]. □

Lemma 8 *Let* Π *be an extended program with signing* S.
Program Π *is consistent iff LB*(Π) $\cap\overline{S}$ *is consistent.*

Proof. Let A^+ be the set $\mathrm{WF}_\perp(\Pi^+)\cup(\mathrm{WF}_\top(\Pi^+)\cap S)$. Note that $A^+\cap\overline{S} = \mathrm{WF}_\perp(\Pi^+)\cap\overline{S}$ and that $A\cap\overline{S} = \mathrm{LB}(\Pi)\cap\overline{S}$. By Theorem 2, we know that A^+ is among the answer sets for Π^+. It is well-known that for all answer sets X of Π^+, $\mathrm{WF}_\perp(\Pi^+)\subset X\subset \mathrm{WF}_\top(\Pi^+)$; and therefore $A^+\cap\overline{S}\subset X\cap\overline{S}$. We know by the definition of a signing for extended programs that for any bad pair $\{b,b'\}\subset \mathrm{atoms}(\Pi^+)$, $\{b,b'\}\subset\overline{S}$. So we can conclude that Π^+ has an answer set that contains no bad pairs iff $A^+\cap\overline{S}$ contains no bad pairs. That is to say, Π^+ has an answer set that contains no bad pairs iff $\mathrm{LB}(\Pi)\cap\overline{S}$ is consistent. By Proposition 2(i), Π is consistent iff Π^+ has an answer set that contains no bad pairs. Therefore, Π is consistent iff $\mathrm{LB}(\Pi)\cap\overline{S}$ is consistent.
□

Lemma 9 *Let* Π *be a consistent extended program with signing* S.
Program Π *entails exactly those literals in* $\overline{S}$ *that are included in LB*(Π).

Proof. Again, let A^+ be the set $\mathrm{WF}_\perp(\Pi^+)\cup(\mathrm{WF}_\top(\Pi^+)\cap S)$. Note that $A = \mathrm{LB}(\Pi)\cup(\mathrm{UB}(\Pi)\cap S)$ and that $A\cap\overline{S} = \mathrm{LB}(\Pi)\cap\overline{S}$. In the proof of Lemma 8 we showed that A^+ contains no bad pairs. Therefore, by Proposition 2(ii), A is among the answer sets for Π. It is easy to show that for all answer sets X of Π, $\mathrm{LB}(\Pi)\subset X$; so $A\cap\overline{S}\subset X\cap\overline{S}$. Thus Π entails exactly those literals in $\overline{S}$ that are included in $\mathrm{LB}(\Pi)\cap\overline{S}$.
□

Lemma 10 *If* Π *is an inconsistent extended program, then program* Π *entails exactly all the literals included in literals*(Π).

Proof. By Proposition 2(iii) and the definition of entailment for extended programs. □

Finally we're ready to prove Theorem 1.

Theorem 1 *For extended programs* P *and* Q *with common signing* S,
if $P_{\overline{S}}\preceq Q_{\overline{S}}$ *and* $Q_S\preceq P_S$ *and literals*$(P)\cap\overline{S}\subset$ *literals*$(Q)\cap\overline{S}$,
then Q *entails every literal in* $\overline{S}$ *that is entailed by* P.

Proof.
Clearly, P^+ and Q^+ are general programs with common signing S. Also, it is easy to see that $P_{\overline{S}} \preceq Q_{\overline{S}} \Rightarrow P^+_{\overline{S}} \preceq Q^+_{\overline{S}}$ and that $Q_S \preceq P_S \Rightarrow Q^+_S \preceq P^+_S$. So by Corollary 1, $\mathrm{WF}_\bot(P^+) \cap \overline{S} \subset \mathrm{WF}_\bot(Q^+) \cap \overline{S}$. Therefore, by the definition of LB, we have $\mathrm{LB}(P) \cap \overline{S} \subset \mathrm{LB}(Q) \cap \overline{S}$.

Consider two cases:

Case 1: Program Q is consistent.

Because Q is a consistent extended program with signing S, we know by Lemma 8 that $\mathrm{LB}(Q) \cap \overline{S}$ is consistent. Since $\mathrm{LB}(P) \cap \overline{S} \subset \mathrm{LB}(Q) \cap \overline{S}$, we know that $\mathrm{LB}(P) \cap \overline{S}$ is also consistent. Therefore, again by Lemma 8, program P is consistent. By Lemma 9, because programs P and Q are consistent, program P entails exactly those literals in $\overline{S}$ that are included in $\mathrm{LB}(P) \cap \overline{S}$, and program Q entails exactly those literals in $\overline{S}$ that are included in $\mathrm{LB}(Q) \cap \overline{S}$. So the theorem is proved for the case when program Q is consistent.

Note that for this case we needn't require that $\mathrm{literals}(P) \cap \overline{S}$ be a subset of $\mathrm{literals}(Q) \cap \overline{S}$.

Case 2: Program Q is not consistent.

In this case, by Lemma 10, program Q entails exactly all the literals included in $\mathrm{literals}(Q)$. So Q entails exactly all the literals in $\overline{S}$ that are in $\mathrm{literals}(Q) \cap \overline{S}$. By the definition of entailment for extended programs, we know that program P can entail at most every literal in $\mathrm{literals}(P)$, so P can entail at most those literals in $\overline{S}$ that are included in $\mathrm{literals}(P) \cap S$. And since $\mathrm{literals}(P) \cap \overline{S} \subset \mathrm{literals}(Q) \cap \overline{S}$, the theorem is proved for the case when program Q is not consistent.
$\square$

Notes

1. This is slightly different from the original definition in [8].

2. In [7], the soundness of πD is proved with respect to the semantics of $\mathcal{A}$ for consistent domain descriptions D *without similar effect-propositions*, where two effect-propositions are *similar* if they differ only in their preconditions. The restriction to domains without similar effect-propositions maintains the soundness of the credit assignment rules (7).

3. For convenience of exposition, we've implicitly assumed a sorted language for $\pi_{\mathrm{forward}}D$; and by "all ground instances" we mean all ground instances of the appropriate sorts in the language of $\pi_{\mathrm{forward}}D$.

4. That is, all ground instances of the appropriate sorts in the language of πD.

5. Dung [2] proved this for a larger class of general programs — the *bottom-stratified & top-strict* programs — which subsumes the signed general programs.

6. The class of order-consistent programs was first defined by Sato in [9], where it is shown that such programs have a consistent completion. Fages' proof in [3] relies crucially on Sato's result in completion semantics. [2] includes a similar, somewhat less general, result.

Acknowledgments

My profound thanks to Vladimir Lifschitz for fundamentally enabling this work. Thanks to Norman McCain for careful reading and comments on a draft of this paper. Thanks also to one of the anonymous referees for several useful, specific criticisms and questions. This work was partially supported by National Science Foundation under grant IRI-9101078.

References

[1] Krzysztof Apt and Marc Bezem. Acyclic programs. In David Warren and Peter Szeredi, editors, *Logic Programming: Proc. of the Seventh Int'l Conf.*, pages 617–633, 1990.

[2] Phan Minh Dung. On the relations between stable and well-founded semantics of logic programs. *Theoretical Computer Science*, 105:7–25, 1992.

[3] François Fages. Consistency of Clark's completion and existence of stable models. Technical report, Ecole Normale Supérieure, 1990. To appear in Methods of Logic in Computer Science.

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

[5] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In David Warren and Peter Szeredi, editors, *Logic Programming: Proc. of the Seventh Int'l Conf.*, pages 579–597, 1990.

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

[7] Michael Gelfond and Vladimir Lifschitz. Representing actions in extended logic programming. In Krzysztof Apt, editor, *Proc. Joint Int'l Conf. and Symp. on Logic Programming*, pages 559–573, 1992.

[8] Kenneth Kunen. Signed data dependencies in logic programs. *Journal of Logic Programming*, 7(3):231–245, 1989.

[9] Taisuke Sato. Completed logic programs and their consistency. *Journal of Logic Programming*, 9:33–44, 1990.

[10] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[11] Allen Van Gelder, Kenneth Ross, and John Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, pages 221–230, 1990.