Language Independence and Language Tolerance in Logic Programs

Norman McCain and Hudson Turner
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712

Abstract

The consequences of a logic program depend in general upon both the rules of the program and its language. However the consequences of some programs are independent of the choice of language, while others depend on the language of the program in only a restricted way. In this paper, we define notions of language independence and language tolerance corresponding to these two cases. Furthermore, we show that there are syntactically-defined classes of programs that are language independent and language tolerant. A primary application of these results is to guarantee that for some programs it is permissible to ignore the fact that the language of the program is many-sorted. This is useful to know, since query evaluation procedures generally take no account of sorts.

1 Introduction

The consequences of a logic program depend in general upon both the rules of the program and its language. For instance, consider the program P_1 whose only rule is

$$p(X) \leftarrow .$$

If the language of P_1 is unsorted and contains, as its only ground terms, the constant symbols a and b, then its consequences are $p(a)$ and $p(b)$. However, if b is replaced by c, then instead its consequences are $p(a)$ and $p(c)$. As a second example, consider the program P_2,

$$\begin{aligned} p(a) &\leftarrow \mathit{not}\ q(X), \\ q(a) &\leftarrow . \end{aligned}$$

Suppose the language of P_2 is the unsorted language that includes only the symbols used in the program; so the only ground term in the language is the constant symbol a. Then $p(a)$ is not a consequence of P_2. However, if the language of P_2 includes in addition the constant symbol b, then $p(a)$ is a consequence of P_2.

Both programs illustrate the fact that the consequences of a program may depend on its language, but the programs differ in one important respect.

In the case of P_1, the consequences of the program taken with respect to any pair of languages inevitably agree in their intersection; for example, wrt the two languages discussed in relation to P_1, the consequences agree on $\{p(a)\}$. However, in the case of P_2, this is not so. For example, $p(a)$ is in the intersection of the languages discussed in relation to P_2, but it is a consequence wrt only one of them. Therefore, P_1 is, in a sense, more tolerant of language changes than P_2. In this paper, we will define a notion of *language tolerance* such that P_1 is language tolerant but P_2 is not.

The consequences of some language tolerant programs do not depend at all on what language is taken as the language of the program.[1] As an example, consider the program P_3,

$$p(X) \leftarrow r(X), \mathit{not}\ q(X),$$
$$r(a) \leftarrow .$$

The consequences of P_3 remain $p(a)$ and $r(a)$, regardless of the language of the program. We will define a notion of *language independence*, generalizing the definition in [Martens and De Schreye, 1992], such that P_3 is language independent but P_1 (and of course, P_2) are not.

Our main purpose in this paper is to identify syntactically-defined classes of programs that are language independent and language tolerant. One reason for desiring such results is suggested in the following paragraph.

In applications of logic programming to knowledge representation, it is often convenient to use a language with many sorts. For instance, a program that represents knowledge about actions in the situation calculus might include such sorts as; *action*, *fluent* and *situation*. Since the standard query evaluation procedures for logic programs take no special account of sorts, this raises a basic question: What is the effect on the declarative meaning of a program when we ignore the fact that its language is many-sorted and replace the language of the program by an unsorted language with the same signature? For a language independent program, it is clear that the consequences are not affected at all, while for a language tolerant program, the result is a conservative extension of the program, with identical consequences in the original many-sorted language.

The question of when it is acceptable to "ignore sorts" in logic programs was the original motivation for these investigations, and it remains the primary application of our results.

In the following section, we specify the syntax and semantics of logic programs. We then define the notion of language independence (Section 3) and a syntactic class of language independent programs (Section 4). We do the same for the notion of language tolerance in Sections 5 & 6. In Section 7 we show that ignoring sorts in a language tolerant program yields

[1] Of course, any candidate language must include the symbols that actually occur in the program, and, if the language is many-sorted, must specify the sorts of these symbols in ways that are compatible with their use.

a conservative extension, and we apply this result to a program for reasoning about actions. We discuss related work in Section 8 and present conclusions in Section 9. Proofs and proof sketches are given in Section 10.

2 Languages, Programs and Answer Sets

To specify a language $\mathcal{L}$ for logic programs, we specify a signature $\Sigma_{\mathcal{L}}$, a nonempty set $I_{\mathcal{L}}$, whose members are called sorts, and a sort specification for each symbol of $\Sigma_{\mathcal{L}}$ as follows.

A *signature* $\Sigma_{\mathcal{L}}$ for a language $\mathcal{L}$ is a triple consisting of disjoint sets of predicate symbols (with arities), function symbols (with arities), and variables. (Constant symbols appear in the signature as zero-ary function symbols.) $\mathcal{L}$ is said to be *one-sorted* (or *unsorted*), if $I_{\mathcal{L}}$ contains exactly one sort. When the language is clear from the context, we may drop the subscript, writing Σ or I.

We assign a *sort specification* to each symbol of Σ as follows. Each variable is assigned a sort in I. Each n-ary function symbol is assigned an $n+1$-tuple $\langle s_1, \ldots, s_n, s_{n+1} \rangle$, where for each i, $1 \leq i \leq n+1$, $s_i \in I$. Each n-ary predicate symbol is assigned an n-tuple $\langle s_1, \ldots, s_n \rangle$, where for each i, $1 \leq i \leq n$, $s_i \in I$. It is stipulated that there must be at least one constant symbol of each sort in I.[2] The terms and atomic formulas (or atoms) of $\mathcal{L}$ are recursively defined in the usual way, respecting the sort specifications of the symbols. Given the atoms, we define the set of literals of $\mathcal{L}$ as the set including, for each atom A, both the atom A and the negated atom $\neg A$. Finally, the rules of $\mathcal{L}$ are the expressions of the form

$$L_1 \mid \ldots \mid L_l \quad \leftarrow \quad L_{l+1}, \ldots, L_m, \ \mathit{not}\ L_{m+1}, \ldots, \ \mathit{not}\ L_n \tag{1}$$

with $0 \leq l \leq m \leq n$, where each L_i $(1 \leq i \leq n)$ is a literal.

A *program* P is a set of rules in a language $\mathcal{L}_P$, which may be one-sorted or many-sorted.[3] The rules of P inevitably belong to a host of other languages as well. These languages may differ from $\mathcal{L}_P$ in their symbols or in their sorts (or both). We will be concerned with how the declarative meaning of a program is affected by choices among these languages.

Definition. Let $\mathcal{L}$ be an arbitrary language, and let P be a program. If every rule in P is a rule of $\mathcal{L}$, we say that $\mathcal{L}$ is *permissible* for P. When the program P is clear from the context, we say simply that $\mathcal{L}$ is permissible.

For every program P, $\mathcal{L}_P$ is permissible for P.

Definition. Given a program P and a language $\mathcal{L}$ that is permissible for P, $\mathcal{H}(P, \mathcal{L})$ is the ground program, consisting of all ground instances, in the

[2] This stipulation is analagous to the stipulation in classical logic that domains are nonempty.

[3] So by a program, unless we say otherwise, we shall mean an extended disjunctive logic program [Gelfond and Lifschitz, 1991].

language $\mathcal{L}$, of rules in P. The language of the ground program $\mathcal{H}(P, \mathcal{L})$ obtained in this manner is $\mathcal{L}$.

In most declarative semantics of logic programs, including the answer set semantics, we take a program P with variables to be essentially a shorthand specification of the ground program $\mathcal{H}(P, \mathcal{L}_P)$, where $\mathcal{L}_P$ is the (often unspecified) language of P.[4] Thus, the answer sets for a program P are the answer sets for the ground program $\mathcal{H}(P, \mathcal{L}_P)$.

In order to finish defining the answer set semantics, we introduce a few more definitions and notational conventions. Given a rule r as in (1), by the subgoals of r we mean

$$L_{l+1}, \ldots, L_m, \ \mathit{not}\ L_{m+1}, \ldots, \ \mathit{not}\ L_n.$$

We define the following: $\mathit{head}(r) = \{L_1, \ldots, L_l\}$, $\mathit{body}(r) = \{L_{l+1}, \ldots, L_n\}$, $\mathit{pos}(r) = \{L_{l+1}, \ldots, L_m\}$, and $\mathit{neg}(r) = \{L_{m+1}, \ldots, L_n\}$. A program P is a *positive program* if, for every rule r in P, $\mathit{neg}(r) = \emptyset$. For a program P, by $\mathit{Lit}(P)$ we denote the set of ground literals that occur in P. For a language $\mathcal{L}$, by $\mathit{Lit}(\mathcal{L})$ we denote the set of ground literals of $\mathcal{L}$.

Now, let P be a positive ground program, with $\mathcal{L}$ a language permissible for P. Let B be a subset of $\mathit{Lit}(\mathcal{L})$. B is *closed under* the rules of P if for all rules r in P, $\mathit{head}(r) \cap B \neq \emptyset$ whenever $\mathit{body}(r) \subset B$. B is *consistent* if B does not contain a pair of complementary literals. B is *logically closed* (wrt $\mathcal{L}$) if either B is consistent or $B = \mathit{Lit}(\mathcal{L})$.

An *answer set* for a positive ground program P is a minimal subset of $\mathit{Lit}(\mathcal{L}_P)$ that is both closed under P and logically closed (wrt $\mathcal{L}_P$).

Let P be a ground program. Let X be a subset of $\mathit{Lit}(\mathcal{L}_P)$. The *reduct* of P wrt X is the positive program P^X, where

$$P^X = \left\{ \begin{array}{c} r' : \exists r \in P \ .\ \mathit{neg}(r) \cap X = \emptyset \wedge \mathit{head}(r') = \mathit{head}(r) \\ \qquad\qquad \wedge \mathit{pos}(r') = \mathit{pos}(r) \wedge \mathit{neg}(r') = \emptyset \end{array} \right\} .$$

Given a ground program P, with $B \subset \mathit{Lit}(\mathcal{L}_P)$, B is an *answer set* for P iff B is an answer set for P^B.

Example. Consider the program P_4,

$$\begin{array}{l} p(X, Y) \ \leftarrow \ r(X), \mathit{not}\ \ q(X, Y), \\ r(a) \ \leftarrow , \\ q(a, 0) \ \leftarrow , \end{array}$$

where the signature $\Sigma_{\mathcal{L}_{P_4}}$ is $(\{p/2, r/1, q/2\}, \{a/0, 0/0, 1/0, 2/0\}, \{X, Y\})$, the set of sorts $I_{\mathcal{L}_{P_4}}$ is $\{\mathit{letter}, \mathit{num}\}$, and the sort specifications for the symbols of $\mathcal{L}_{P_4}$ are $\mathit{sort}(X) = \mathit{letter}$, $\mathit{sort}(Y) = \mathit{num}$, $\mathit{sort}(p) = \mathit{sort}(q) =$

[4] By convention, when the language $\mathcal{L}_P$ of a program P is not otherwise specified, we assume that $\mathcal{L}_P$ is the minimal unsorted language that (i) is permissible for P, and (ii) includes the constant symbol a if no constant symbol occurs in P.

$\langle letter, num\rangle$, $sort(r) = \langle letter\rangle$, $sort(a) = \langle letter\rangle$, and $sort(0) = sort(1) = sort(2) = \langle num\rangle$.

The ground program $\mathcal{H}(P_4, \mathcal{L}_{P_4})$ is

$$\begin{aligned} p(a,0) &\leftarrow r(a),\ not\ q(a,0),\\ p(a,1) &\leftarrow r(a),\ not\ q(a,1),\\ p(a,2) &\leftarrow r(a),\ not\ q(a,2),\\ r(a) &\leftarrow ,\\ q(a,0) &\leftarrow . \end{aligned}$$

Let $A = \{p(a,1), p(a,2), r(a), q(a,0)\}$. The reduct P_4^A is the positive program

$$\begin{aligned} p(a,1) &\leftarrow r(a),\\ p(a,2) &\leftarrow r(a),\\ r(a) &\leftarrow ,\\ q(a,0) &\leftarrow . \end{aligned}$$

Since A is a minimal set that is both closed under the rules of P_4^A and logically closed, A is an answer set for P_4. It is, in fact, the only answer set, and of course it is consistent.

A program P *entails* exactly those literals from $Lit(\mathcal{L}_P)$ that are included in every answer set for P. We denote the set of literals entailed by P by $Cn(P)$. A program P is *consistent* if $Cn(P)$ is consistent, and *inconsistent* otherwise.

We make two simple observations: (i) for a program P, $Cn(P)$ is the set of literals from $Lit(\mathcal{L}_P)$ that are included in every consistent answer set for P, and (ii) if a program P is inconsistent, then either it has no answer set or its only answer set is $Lit(\mathcal{L}_P)$.

We will at times be interested in the class of normal programs. A program P is a *normal* program if every rule in P has the form

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

with $0 \leq m \leq n$, where all A_i $(i = 0, \ldots, n)$ are atoms of $\mathcal{L}_P$.

3 Language Independence

In this section we define the notion of language independence and state a simple proposition regarding the consequences of language independent programs.

Definition. A program P is *language independent* if, for any two languages $\mathcal{L}_1, \mathcal{L}_2$ that are permissible for P, the ground programs $\mathcal{H}(P, \mathcal{L}_1)$ and $\mathcal{H}(P, \mathcal{L}_2)$ have the same consistent answer sets.

Proposition 3.1 *Let P be a language independent program, and let $\mathcal{L}_1, \mathcal{L}_2$ be permissible languages for P. Then $Cn(\mathcal{H}(P, \mathcal{L}_1)) = Cn(\mathcal{H}(P, \mathcal{L}_2))$.*

Ground programs are trivially language independent. Let P be a ground program. For any language $\mathcal{L}$ that is permissible for P, we have $P = \mathcal{H}(P, \mathcal{L})$. Hence, for any two languages $\mathcal{L}_1, \mathcal{L}_2$ that are permissible for P, $\mathcal{H}(P, \mathcal{L}_1) = \mathcal{H}(P, \mathcal{L}_2)$. So P is language independent. Thus, for the purpose of determining answer sets and consequences, there is no need to specify the language of a consistent ground program.

We will be interested in identifying additional classes of language independent and language tolerant programs.

4 Allowed Programs are Language Independent

In this section we present a theorem which states that *allowed* programs are language independent. The class of allowed programs is syntactically-defined and was studied in [Lloyd and Topor, 1986] in connection with the problem of floundering in SLDNF. The same class of programs is also known as *range-restricted*.

We begin by generalizing the definition of an allowed program, which was originally defined for normal programs only.

Definition. Let P be a program. A rule $R \in P$ is *allowed* if every variable in R occurs in $pos(R)$. The program P is *allowed* if every rule in P is allowed.

The program P_3 is allowed, but the programs P_1, P_2 and P_4 are not. We can now state the following theorem.

Theorem 4.1 *Every allowed program is language independent.*

The theorem shows that for allowed programs, as for ground programs, consistent answer sets are unaffected by the choice of language. However, the reason for this is different in the case of allowed programs, since the choice of language $\mathcal{L}$ for an allowed program P generally does affect $\mathcal{H}(P, \mathcal{L})$. As an example, suppose that $\mathcal{L}_1$ and $\mathcal{L}_2$ are permissible languages for P_3, with a as the only ground term in $\mathcal{L}_1$, and a and b as the only ground terms in $\mathcal{L}_2$. Then, although the ground program $\mathcal{H}(P_3, \mathcal{L}_1)$ is

$$\begin{aligned} p(a) &\leftarrow r(a),\ not\ q(a), \\ r(a) &\leftarrow , \end{aligned}$$

and the ground program $\mathcal{H}(P_3, \mathcal{L}_2)$ is

$$\begin{aligned} p(a) &\leftarrow r(a),\ not\ q(a), \\ p(b) &\leftarrow r(b),\ not\ q(b), \\ r(a) &\leftarrow , \end{aligned}$$

the answer sets and consequences for the two programs are identical. Theorem 4.1 implies that the same result holds for all allowed programs and their permissible languages.

5 Language Tolerance

In this section we define the notion of language tolerance and state a simple proposition regarding the consequences of language tolerant programs.

Definition. A program P is *language tolerant* if, for any two languages $\mathcal{L}_1, \mathcal{L}_2$ that are permissible for P the following holds: If A_1 is a consistent answer set for the ground program $\mathcal{H}(P, \mathcal{L}_1)$, then there is a consistent answer set A_2 for the ground program $\mathcal{H}(P, \mathcal{L}_2)$ s.t. $A_1 \cap Lit(\mathcal{L}_2) = A_2 \cap Lit(\mathcal{L}_1)$.

Proposition 5.1 *Let P be a language tolerant program, and let $\mathcal{L}_1, \mathcal{L}_2$ be permissible languages for P. Then $Cn(\mathcal{H}(P, \mathcal{L}_1)) \cap Lit(\mathcal{L}_2) = Cn(\mathcal{H}(P, \mathcal{L}_2)) \cap Lit(\mathcal{L}_1)$.*

Let $X = Lit(\mathcal{L}_1) \cap Lit(\mathcal{L}_2)$. Since the answer sets for $\mathcal{H}(P, \mathcal{L}_1)$ are subsets of $Lit(\mathcal{L}_1)$ and the answer sets for $\mathcal{H}(P, \mathcal{L}_2)$ are subsets of $Lit(\mathcal{L}_2)$, the equality expressions in the previous definition and proposition can be replaced by $A_1 \cap X = A_2 \cap X$ and $Cn(\mathcal{H}(P, \mathcal{L}_1)) \cap X = Cn(\mathcal{H}(P, \mathcal{L}_2)) \cap X$, respectively. Thus, if P is language tolerant, the consequences of P, determined wrt any pair of permissible languages, agree in the intersection of the languages.

Clearly, every language independent program is language tolerant, but the converse does not hold. This is illustrated by the programs P_1 and P_4. It is easy to show that these programs are not language independent.

Example. [cont.] Let $\mathcal{L}$ be the language that differs from $\mathcal{L}_{P_4}$ by including an additional constant symbol b of sort *letter* and by replacing the constant symbol 1 in $\mathcal{L}_{P_4}$ by 4. The ground program $\mathcal{H}(P_4, \mathcal{L})$, which contains eight rules, has the unique answer set $\{p(a,2), p(a,4), r(a), q(a,0)\}$. The unique answer set for $\mathcal{H}(P_4, \mathcal{L}_{P_4})$ is $\{p(a,1), p(a,2), r(a), q(a,0)\}$, as given in Section 2. Since these answer sets differ, P_4 is not language independent. However, the two answer sets agree in the the intersection of the two languages.

It will be easy to show that P_1 and P_4 are language tolerant, using the results obtained in the next section.

6 Some Stable Programs are Language Tolerant

The class of stable programs [Stroetman, 1993] properly includes the class of allowed programs. Under a rather strong restriction, we can show that stable programs are language tolerant. The restriction is stated after the following definition.

Definition. A program Q is a *part* of a program P if Q can be obtained from P by (i) selecting a subset of the rules in P and (ii) deleting zero or more subgoals from each selected rule.

We can show that a stable program is language tolerant if, for every language $\mathcal{L}$ that is permissible for P, every part of $\mathcal{H}(P, \mathcal{L})$ has a consistent

answer set. In general of course it may be difficult determine whether a program has this property, but there are some easily recognized classes of programs that do; for instance, stratified normal programs. Later in this section we will define a larger class of normal programs with this property, namely, the class of predicate-order-consistent programs.

We begin our discussion of stable programs by generalizing a number of definitions given originally in [Stroetman, 1993] in the framework of normal programs.

An I/O specification σ for a program P is a function that maps every n-ary predicate symbol Q that occurs in P to a pair of modes — σ_Q and $\sigma_{\neg Q}$ — each of which is a function from the set $\{1, \ldots, n\}$ to $\{+, -\}$. [5]

The mode σ_Q can be conveniently written as $Q(\sigma_Q(1), \ldots, \sigma_Q(n))$. For example, $\neg Holds(-, +)$ means that $\sigma_{\neg Holds}(1) = -$ and $\sigma_{\neg Holds}(2) = +$.

Definition. Let P be a program with I/O specification σ and permissible language $\mathcal{L}$. For any expression E from $\mathcal{L}$, by $FV(E)$ we designate the set of free variables that occur in E. For any atom $Q(t_1, \ldots, t_n)$ in $\mathcal{L}$ s.t. Q occurs in P,

$$\begin{aligned} FV^+(Q(t_1, \ldots, t_n)) &= \bigcup\{FV(t_i) : i \in \{1, \ldots, n\} \text{ and } \sigma_Q(i) = +\} \\ FV^-(Q(t_1, \ldots, t_n)) &= \bigcup\{FV(t_i) : i \in \{1, \ldots, n\} \text{ and } \sigma_Q(i) = -\} \ . \end{aligned}$$

For any negated atom $\neg Q(t_1, \ldots, t_n)$ in $\mathcal{L}$, s.t. Q occurs in P,

$$\begin{aligned} FV^+(\neg Q(t_1, \ldots, t_n)) &= \bigcup\{FV(t_i) : i \in \{1, \ldots, n\} \text{ and } \sigma_{\neg Q}(i) = +\} \\ FV^-(\neg Q(t_1, \ldots, t_n)) &= \bigcup\{FV(t_i) : i \in \{1, \ldots, n\} \text{ and } \sigma_{\neg Q}(i) = -\} \ . \end{aligned}$$

Definition. Let P be a program. A rule $R \in P$ is *stable wrt* an I/O specification σ if there exists an ordering $L_1, \ldots, L_k$ of $pos(R)$ such that at least one of the following conditions is satisfied for every variable x that occurs in R:

(i) $head(R) \neq \emptyset \wedge \forall L \in head(R)[x \in FV^+(L)]$,

(ii) $\exists i \in \{1, \ldots, k\}[x \in FV^-(L_i) \wedge \forall j \in \{1, \ldots, i\}[x \notin FV^+(L_j)]]$.

Program P is *stable wrt* σ if every rule in P is stable wrt σ, and P is *stable* if for some σ, P is stable wrt σ.

The programs P_1, P_3, and P_4 are stable. P_1 is stable wrt the I/O specification $p(+)$. P_3 is stable wrt $p(-)$, $q(-)$, and $r(-)$. P_4 is stable wrt $p(-, +)$, $q(-, -)$, and $r(-)$. The program P_2 is not stable.

[5] Intuitively, if $\sigma_Q(i) = +$, then i is an input position for any atom with the predicate Q, and if $\sigma_Q(i) = -$, then i is an output position. Similarly, if $\sigma_{\neg Q}(i) = +$, then i is an input position for any negated atom with the predicate Q, and if $\sigma_{\neg Q}(i) = -$, then i is an output position. An input position is an argument place that should be a ground term in any call to Q. An output position need not be ground. These concepts are associated with the procedural semantics of normal logic programs.

It is easy to see that a program P is allowed iff it is stable wrt the I/O specification that maps every argument place to $-$. So the class of stable programs is a superset of the class of allowed programs.

The preceding definition generalizes the definition in [Stroetman, 1993] to the class of extended disjunctive programs. But even in the special case of normal programs, the definition is more general. In [Stroetman, 1993] the body of a rule is an ordered sequence rather than a set, and whether or not a rule is stable may depend on this ordering. For instance, according to the original definition, the rule $p \leftarrow \mathit{not}\ r(X), q(X)$ is not stable. By the definition given here, on the other hand, the rule is stable wrt any I/O specification σ such that $\sigma_q(1) = -$.

It is not the case that every stable program is language tolerant, as the following program P_5 illustrates,

$$\begin{aligned} p(X) &\leftarrow \mathit{not}\ p(X), \\ p(a) &\leftarrow . \end{aligned}$$

P_5 is stable wrt the I/O specification $p(+)$. Suppose that $\mathcal{L}_1$ is the minimal unsorted permissible language for P_5 and that $\mathcal{L}_2$ is the same as $\mathcal{L}_1$ except that it contains the additional constant symbol b. Then $\mathcal{H}(P_5, \mathcal{L}_1)$ has a single answer set $\{p(a)\}$, but $\mathcal{H}(P_5, \mathcal{L}_2)$ is inconsistent and has no answer sets at all. This shows that P_5 is not language tolerant.

It is also not the case that every stable program that is consistent wrt every permissible language is language tolerant. This is illustrated by the program P_6.

$$\begin{aligned} p(X) &\leftarrow d, \mathit{not}\ p(X), \\ p(a) &\leftarrow , \\ c \mid d &\leftarrow . \end{aligned}$$

P_6 is stable wrt the I/O specification $p(+)$, and for every permissible language $\mathcal{L}$ for P_6, $\{p(a), c\}$ is an answer set for $\mathcal{H}(P_6, \mathcal{L})$. Suppose that $\mathcal{L}_1$ is the minimal unsorted permissible language for P_6 and that $\mathcal{L}_2$ is the same as $\mathcal{L}_1$ except that it contains the additional constant symbol b. Then $\mathcal{H}(P_6, \mathcal{L}_1)$ has two answer sets $\{p(a), c\}$ and $\{p(a), d\}$, but $\mathcal{H}(P_6, \mathcal{L}_2)$ has only the answer set $\{p(a), c\}$. This shows that P_6 is not language tolerant and motivates the stronger condition stated in the following theorem.

Theorem 6.1 *If a program P is stable and, for every permissible language $\mathcal{L}$ for P, every part of $\mathcal{H}(P, \mathcal{L})$ has a consistent answer set, then P is language tolerant.*

As given, Theorem 6.1 is difficult to apply because of the consistency condition in the statement of the theorem. We now turn to the problem of defining a general syntactic class of programs that satisfy this condition. For this purpose, we define the property of predicate-order-consistency in

a manner analogous to the definition of order-consistency in [Fages, 1993]. Unlike the definition of order-consistency, the definition of predicate-order-consistency does not refer to the language of the program, but only to the predicate symbols that occur in the program.

Let P be a normal program. The property of predicate-order-consistency is defined in terms of the *predicate dependency graph* $G(P)$ of P. The nodes of the graph are the predicate symbols that occur in P.

Let p, q be predicate symbols that occur in P. There is a positive edge in $G(P)$ from p to q if there is a rule $R \in P$ with p occurring in $pos(R)$ and q occurring in $head(R)$, and there is a negative edge in $G(P)$ from p to q if there a rule $R \in P$ with p occurring in $neg(R)$ and q occurring in $head(R)$.

Definition. Given $G(P)$, we define two relations. For all predicate symbols p, q that occur in P,

(i) $p \leq_+ q$ if there is a path in $G(P)$ from p to q with an even number of negative edges,

(ii) $p \leq_- q$ if there is a path in $G(P)$ from p to q with an odd number of negative edges.

Finally, a third relation $\leq_\pm$ is defined in terms of the previous two: For all predicate symbols p, q that occur in P, $p \leq_\pm q \equiv [p \leq_+ q \text{ and } p \leq_- q]$.

Definition. A normal program P is *predicate-order-consistent* if the relation $\leq_\pm$ in $G(P)$ is well-founded and there is no predicate symbol p that occurs in P s.t. $p \leq_\pm p$.

The following proposition is proved by using the result from [Fages, 1993] which states that every order-consistent normal program has an answer set.

Proposition 6.1 *If P is a predicate-order-consistent normal program, then for every permissible language $\mathcal{L}$ for P, every part of $\mathcal{H}(P, \mathcal{L})$ has a consistent answer set.*

Theorem 6.2 *If normal program P is stable and predicate-order-consistent, then P is language tolerant.*

Since the programs P_1 and P_4 clearly satisfy the conditions of Theorem 6.2, they are language tolerant.

7 Ignoring Sorts

In this section, we define the notion of "ignoring sorts" and state a theorem that justifies ignoring sorts in language tolerant programs.

Definition. Let P and P' be ground programs such that $P \subset P'$. We say that P' is a *conservative extension* of P if the following condition holds: A is a consistent answer set for P iff there is a consistent answer set A' for P' such that $A = A' \cap Lit(\mathcal{L}_P)$.

Proposition 7.1 *If a program P' is a conservative extension of a program P, then $Cn(P) = Cn(P') \cap Lit(\mathcal{L}_P)$.*

Definition. Let $\mathcal{L}, \mathcal{L}'$ be languages. We say that $\mathcal{L}'$ is obtained from $\mathcal{L}$ by *ignoring sorts* if $\Sigma_{\mathcal{L}} = \Sigma_{\mathcal{L}'}$ and $\mathcal{L}'$ is one-sorted.

Proposition 7.2 *Let P be a language tolerant program. If $\mathcal{L}$ is obtained from $\mathcal{L}_P$ by ignoring sorts, then $\mathcal{H}(P, \mathcal{L})$ is a conservative extension of $\mathcal{H}(P, \mathcal{L}_P)$.*

We now apply Proposition 7.2 to a program for reasoning about actions from [Lifschitz *et al.*, 1993]. The language of the program is a many-sorted language with four sorts: *fluent*, *action*, *truth-value*, and *situation*. The language contains the following constant symbols: *Loaded* and *Alive* of sort *fluent*; *Load*, *Wait*, and *Shoot* of sort *action*; 0 and 1 of sort *truth-value*; and $S0$ of sort *situation*. In addition, the language contains the function symbol *Result*, where

$$sort(Result) \;=\; \langle action, situation, situation \rangle \ ,$$

and the predicate symbols, *Holds*, *Holds'*, *Noninertial*, and *Rel*, where

$$\begin{aligned} sort(Holds) \;&=\; sort(Holds') \;=\; \langle fluent, situation \rangle \\ sort(Noninertial) \;&=\; \langle fluent, action, situation, truth\text{-}value \rangle \\ sort(Rel) \;&=\; \langle situation \rangle \ . \end{aligned}$$

The sorts of the variables, f, a, and s, can be inferred from their use.

The program P_7 is the *positive form* of an extended program given in [Lifschitz *et al.*, 1993].[6] It formalizes the so-called Murder Mystery variant of the Yale Shooting Problem [Hanks and McDermott, 1987].

1. $Holds(Alive, S0)$.

2. $Holds'(Alive, Result(Wait, Result(Shoot, S0)))$.

2a. $Rel(Result(Wait, Result(Shoot, S0)))$.

2b. $Rel(Result(Shoot, S0))$.

3. $Holds(Loaded, Result(Load, s))$.

4. $Noninertial(Loaded, Load, s, 1)$.

5. $Holds'(Alive, Result(Shoot, s)) \leftarrow Holds(Loaded, s)$.

6. $Noninertial(Alive, Shoot, s, 0) \leftarrow \ not \ \ Holds'(Loaded, s)$.

7. $Holds'(Loaded, s) \leftarrow Rel(Result(Shoot, s)), Holds(Alive, Result(Shoot, s))$.

[6] The positive form was obtained by replacing each literal of the form $\neg Holds(\_, \_)$ by an atom of the form $Holds'(\_, \_)$, where $Holds'$ is a new predicate symbol. See [Gelfond and Lifschitz, 1991].

8. $Holds(Loaded, s) \leftarrow Rel(Result(Shoot, s)), Holds(Alive, s),$
$Holds'(Alive, Result(Shoot, s))$.

9. $Holds'(Loaded, Result(Shoot, s))$.

10. $Noninertial(Loaded, Shoot, s, 0)$.

11. $Holds(f, Result(a, s)) \leftarrow Holds(f, s),\ not\ \ Noninertial(f, a, s, 0)$.

12. $Holds'(f, Result(a, s)) \leftarrow Holds'(f, s),\ not\ \ Noninertial(f, a, s, 1)$.

13. $Holds(f, s) \leftarrow Rel(Result(a, s)), Holds(f, Result(a, s)),$
$not\ \ Noninertial(f, a, s, 1)$.

14. $Holds'(f, s) \leftarrow Rel(Result(a, s)), Holds'(f, Result(a, s)),$
$not\ \ Noninertial(f, a, s, 0)$.

The program is not allowed, because in each of the rules 3, 4, 6, 9, 10, 11 and 12 there is a variable that does not occur in the positive part of the body.

Let σ be the following I/O specification: $Holds(-,+)$, $Holds'(-,+)$, $Noninertial(+,+,+,+)$, and $Rel(-)$. It is easy to check that the program is stable wrt σ. Also, it is easy to see that the program is predicate-order-consistent; in fact, the relation $\leq_{\pm}$, which is defined wrt the predicate dependency graph $G(P_7)$, is empty. Thus, by Theorem 6.2, P_7 is language tolerant.

Let $\mathcal{L}'_{P_7}$ be the language that is obtained from $\mathcal{L}_{P_7}$ by ignoring sorts. By Proposition 7.2, $\mathcal{H}(P_7, \mathcal{L}'_{P_7})$ is a conservative extension of $\mathcal{H}(P_7, \mathcal{L}_{P_7})$. So ignoring sorts in $\mathcal{L}_{P_7}$ has no effect on whether or not an atom in the original many-sorted language is a consequence of the program. For P_7, this justifies the use of query evaluation procedures that take no account of sorts.

8 Discussion

A notion of *language independence* is defined for stratified normal programs in [Martens and De Schreye, 1992] as follows: "A stratified program P with underlying language $\mathcal{L}_P$ is called *language independent* iff for any extension $\mathcal{L}'$ for $\mathcal{L}_P$, its perfect $\mathcal{L}'$-Herbrand model is equal to its perfect $\mathcal{L}_P$-Herbrand model." (Here $\mathcal{L}_P$ is the minimal (unsorted) permissible language for P.) Furthermore, the following result is stated as Proposition 2.5: "Let P be a stratified program. If P is range-restricted then P is language-independent." Note that the class of allowed programs and the class of range-restricted programs are the same.

Since the perfect model semantics and the answer set semantics agree for the class of stratified normal programs, it is possible to compare the preceding proposition with our Theorem 4.1. First, our theorem applies to sorted as well as unsorted languages. Secondly, it applies to non-stratified normal programs and more generally to the entire class of extended disjunctive programs. In these respects, Theorem 4.1 is more general than Proposition 2.5.

Since our notion of language independence considers all pairs of permissible languages, not only the minimal unsorted permissible language and each of its unsorted extensions, Theorem 4.1 is at least as strong as Proposition 2.5. There is no definition in [Martens and De Schreye, 1992] analogous to our notion of language tolerance.

To our knowledge, the closest analogues to our definition of language tolerance appear in [Topor and Sonenberg, 1988] and [Ross, 1993]. In [Topor and Sonenberg, 1988], a concept called "domain independence" is defined as follows: "A stratified database D is *domain independent* if, for all languages $\mathcal{L}_1$ and $\mathcal{L}_2$ extending that of D, and for all atoms A in $\mathcal{L}_1$ and $\mathcal{L}_2$, $ans(A, D, \mathcal{L}_1) = ans(A, D, \mathcal{L}_2)$." (Here $ans(A, D, \mathcal{L})$ may be taken to be the set of ground instances of A that are consequences of $\mathcal{H}(D, \mathcal{L})$ according to any of the various semantics that coincide on stratified programs.) This definition resembles our definition of language tolerance in its focus on the intersections of pairs of languages. However, sorted languages and extended or disjunctive programs are not covered, and the largest syntactic class of programs shown to be domain independent is the class of allowed stratified programs.

In [Ross, 1993], definitions and results bearing a family resemblance to ours are presented, but in the framework of HiLog languages rather than sorted first-order languages. While it is clear that these ideas are related to our notions of language tolerance and stability, we are not yet able to describe the relationships precisely. This is a topic for further study.

9 Conclusion

The classes of allowed and stable programs have been previously studied in connection with the problem of floundering in SLDNF. We have generalized these classes to include extended disjunctive programs and shown results relating them to the notions of language independence and language tolerance. We have applied these results to show that the practice of "ignoring sorts" when evaluating queries wrt a logic program in a many-sorted language can sometimes be justified declaratively, in the sense that the program that results from ignoring sorts is a conservative extension of the original program.

It is interesting to note that the class of stable programs does not include all positive programs, which intuitively are also language tolerant. It should be possible, therefore, to find yet larger classes of programs that are language tolerant. This is a topic for further study.

10 Proofs

Definition. Given a ground program P, a permissible language $\mathcal{L}$, and a set $X \subset Lit(\mathcal{L})$, we say that a rule $r \in P$ is *confined to* X if either $pos(r) \not\subset X$

or $head(r) \subset X$. We say that P is *confined to* X, if every rule in P is confined to X.

Definition. For ground program P and $X \subset Lit(P)$, let $c_X(P) = \{r \in P : pos(r) \subset X\}$.

Proposition 10.1 *If ground program P is confined to X, then the consistent answer sets for P are the consistent answer sets for $c_X(P)$.*

The proof of this proposition is straightforward. The following lemma is asserted without proof. Note that $FV(L)$ denotes the set of free variables in the literal L, as defined in Section 6.

Lemma 10.1 *Let $\mathcal{L}_1, \mathcal{L}_2$ be languages. If $A_1, \ldots, A_k, B$ are literals of $\mathcal{L}_1$ and $\mathcal{L}_2$, θ is a substitution in $\mathcal{L}_1$, and*

$$FV(B) \subset \bigcup_{i=1}^{k} FV(A_i),$$

then if $A_1\theta, \ldots, A_k\theta \in Lit(\mathcal{L}_2)$ then $B\theta \in Lit(\mathcal{L}_2)$.

Lemma 10.2 *Let P be an allowed program with permissible languages $\mathcal{L}_1$ and $\mathcal{L}_2$. Let $X = Lit(\mathcal{L}_1) \cap Lit(\mathcal{L}_2)$. Programs $\mathcal{H}(P, \mathcal{L}_1)$ and $\mathcal{H}(P, \mathcal{L}_2)$ are each confined to X.*

Proof. Suppose $r \in \mathcal{H}(P, \mathcal{L}_1)$. Then there is a rule $R \in P$ and a substitution θ in $\mathcal{L}_1$ s.t. $r = R\theta$. We will show that if $pos(r) \subset X$ then $head(r) \subset X$. Suppose $pos(r) \subset X$. Clearly, $head(r) \subset Lit(\mathcal{L}_1)$. It remains to show that $head(r) \subset Lit(\mathcal{L}_2)$. So, suppose $L \in head(r)$. Then for some literal B in $head(R)$, $L = B\theta$. Let $A_1, \ldots, A_k$ be the literals in $pos(R)$. Since $\mathcal{L}_1, \mathcal{L}_2$ are permissible, $A_1, \ldots, A_k, B$ are literals of $\mathcal{L}_1, \mathcal{L}_2$. Since $r \in \mathcal{H}(P, \mathcal{L}_1)$, $A_1\theta, \ldots, A_k\theta \in Lit(\mathcal{L}_1)$. Since R is allowed,

$$FV(B) \subset \bigcup_{i=1}^{k} FV(A_i).$$

Since $A_1\theta, \ldots, A_k\theta \in pos(r) \subset X \subset Lit(\mathcal{L}_2)$, we conclude by Lemma 10.1 that $B\theta \in Lit(\mathcal{L}_2)$. It follows that $head(r) \subset Lit(\mathcal{L}_2)$. So $\mathcal{H}(P, \mathcal{L}_1)$ is confined to X. By symmetry, $\mathcal{H}(P, \mathcal{L}_2)$ is confined to X. □

Lemma 10.3 *Let P be a program with permissible languages $\mathcal{L}_1$ and $\mathcal{L}_2$. Let $X = Lit(\mathcal{L}_1) \cap Lit(\mathcal{L}_2)$. If P is allowed, then $c_X(\mathcal{H}(P, \mathcal{L}_1)) = c_X(\mathcal{H}(P, \mathcal{L}_2))$.*

Proof. Suppose $r \in c_X(\mathcal{H}(P, \mathcal{L}_1))$. Then $pos(r) \subset X$. By Lemma 10.2, $\mathcal{H}(P, \mathcal{L}_1)$ is confined to X, so $head(r) \subset X$. It remains to show that $neg(r) \subset X$. This is proved, using Lemma 10.1, in a manner similar to that used in proving the previous lemma. So $head(r) \cup body(r) \subset X \subset Lit(\mathcal{L}_2)$. So $r \in \mathcal{H}(P, \mathcal{L}_2)$. Since $pos(r) \subset X$, $r \in c_X(\mathcal{H}(P, \mathcal{L}_2))$. Thus, $c_X(\mathcal{H}(P, \mathcal{L}_1)) \subset c_X(\mathcal{H}(P, \mathcal{L}_2))$. By symmetry, $c_X(\mathcal{H}(P, \mathcal{L}_2)) \subset c_X(\mathcal{H}(P, \mathcal{L}_1))$. □

Proof. (of Theorem 4.1) Suppose P is an allowed program. To show that P is language independent, we show that for any two permissible languages $\mathcal{L}_1, \mathcal{L}_2$, $\mathcal{H}(P, \mathcal{L}_1)$ and $\mathcal{H}(P, \mathcal{L}_2)$ have the same consistent answer sets. Let $X = Lit(\mathcal{L}_1) \cap Lit(\mathcal{L}_2)$. By Lemma 10.3, $c_X(\mathcal{H}(P, \mathcal{L}_1)) = c_X(\mathcal{H}(P, \mathcal{L}_2))$. By Proposition 10.1, the consistent answer sets for $\mathcal{H}(P, \mathcal{L}_1)$ are the consistent answer sets for $c_X(\mathcal{H}(P, \mathcal{L}_1))$, and the consistent answer sets for $\mathcal{H}(P, \mathcal{L}_2)$ are the consistent answer sets for $c_X(\mathcal{H}(P, \mathcal{L}_2))$. It follows that $\mathcal{H}(P, \mathcal{L}_1)$ and $\mathcal{H}(P, \mathcal{L}_2)$ have the same consistent answer sets. Therefore, P is language independent. □

The following definitions and theorem from [Lifschitz and Turner, 1994] are used in the proof of Theorem 6.1.

Definition. Given a rule r, $lit(r)$ stands for $head(r) \cup body(r)$. A *splitting set* for a ground program P is any set U of ground literals such that, for every rule $r \in P$, if $head(r) \cap U$ is nonempty then $lit(r) \subset U$. The set of rules $r \in P$ such that $lit(r) \subset U$ is called the *bottom* of P relative to the splitting set U and denoted by $b_U(P)$. The set $t_U(P)$ is the *top* of P relative to U. Consider two sets of literals U, X and a program P. For each rule $r \in P$ such that $pos(r) \cap U$ is a subset of X and $neg(r) \cap U$ is disjoint from X, take the rule r' defined by

$$head(r') = head(r),\ pos(r') = pos(r) \setminus U,\ neg(r') = neg(r) \setminus U.$$

The program consisting of all rules r' obtained in this way will be denoted by $e_U(P, X)$. Let U be a splitting set for a program P. A *solution* to P (with respect to U) is a pair $\langle X, Y \rangle$ of sets of literals such that

- X is an answer set for $b_U(P)$,
- Y is an answer set for $e_U(t_U(P), X)$,
- $X \cup Y$ is consistent.

Splitting Set Theorem [Lifschitz and Turner, 1994]. *Let U be a splitting set for a program P. A set A of literals is a consistent answer set for P if and only if $A = X \cup Y$ for some solution $\langle X, Y \rangle$ to P with respect to U.*

We prove the following corollary to the Splitting Set Theorem.

Corollary 10.4 *Let P be a ground program with splitting set U s.t. every literal in U has its complement in U. If every part of P has a consistent answer set, then the sets $\{A \cap U : A$ is a consistent answer set for $P\}$ and $\{B : B$ is a consistent answer set for $b_U(P)\}$ coincide.*

Proof. Left-to-right follows immediately from the Splitting Set Theorem. To see the other direction, assume that B is a consistent answer set for $b_U(P)$. We must show that there is a consistent answer set A for P s.t. $A \cap U = B$. Since every part of P has a consistent answer set, and since $e_U(t_U(P), B)$ is a part of P, $e_U(t_U(P), B)$ has a consistent answer set. Let C

be a consistent answer set for $e_U(t_U(P), B)$. Let $A = B \cup C$. We need to show that A is a consistent set. We know that every consistent answer set for a program is a subset of the literals in the program. So, since $Lit(b_U(P)) \subset U$ and $Lit(e_U(t_U(P), X)) \subset Lit(\mathcal{L}_P) \setminus U$, we conclude that $B \subset U$ and $C \subset Lit(\mathcal{L}_P) \setminus U$. Because every literal in U has its complement in U, we know that every literal in B has its complement in U, and it follows that no literal in B has its complement in C. So $B \cup C = A$ is a consistent set, and we have shown that $\langle B, C \rangle$ is a solution to P with respect to U. By the Splitting Set Theorem we conclude that A is a consistent answer set for P. Furthermore, $A \cap U = B$. $\Box$

Definition. A ground program P is *stable wrt* (U, X), $U \subset X \subset Lit(\mathcal{L}_P)$, if for every rule $r \in P$, at least one of the following three conditions holds:

(i) $head(r) \cup body(r) \subset U$

(ii) $head(r) \subset X \setminus U$

(iii) $pos(r) \not\subset X$.

Lemma 10.5 *A ground program P is stable wrt (U, X) iff P is confined to X and U splits $c_X(P)$.*

Proof. To prove the left-to-right direction, suppose P is stable wrt (U, X). Let r be a rule in P. If r satisfies condition (iii) then r is trivially confined to X. If r satisfies condition (i) or (ii) then $head(r) \subset X$, so again r is confined to X. So P is confined to X. Every rule in $c_X(P)$ satisfies either condition (i) or (ii). These rules are clearly split by U. To prove the right-to-left direction, suppose P is confined to X and U splits $c_X(P)$. We must show that each rule in P satisfies one of the three conditions in the definition of stable wrt (U, X). Suppose r is a rule in P. If r satisfies condition (iii), we are done. So suppose it does not. Then since P is confined to X, $head(r) \subset X$. Since U splits $c_X(P)$, either $head(r) \cap U = \emptyset$ or $head(r) \cup body(r) \subset U$. If $head(r) \cap U = \emptyset$ then r satisfies condition (ii). If $head(r) \cup body(r) \subset U$ then r satisfies condition (i). $\Box$

Proposition 10.2 *Let P be a ground program. Let $U \subset X \subset Lit(\mathcal{L}_P)$ s.t. every literal in U has its complement in U. If P is stable wrt (U, X), and if every part of P has a consistent answer set, then B is a consistent answer set for $b_U(P)$ iff there is a consistent answer set A for P s.t. $A \cap U = B$.*

Proof. Suppose P is stable wrt (U, X) and that every part of P has a consistent answer set. By Lemma 10.5, P is confined to X and U splits $c_X(P)$. Since $c_X(P) \subset P$, every part of $c_X(P)$ has a consistent answer set. By Corollary 10.4, the sets $\{A \cap U : A \text{ is a consistent answer set for } c_X(P)\}$ and $\{B : B \text{ is a consistent answer set for } b_U(c_X(P))\}$ coincide. Furthermore, $b_U(c_X(P)) = b_U(P)$, since

$$b_U(c_X(P)) = \{r \in c_X(P) : head(r) \cup body(r) \subset U\}$$

$$
\begin{aligned}
&= \{r \in P : pos(r) \subset X \wedge head(r) \cup body(r) \subset U\} \\
&= \{r \in P : head(r) \cup body(r) \subset U\} \\
&= b_U(P).
\end{aligned}
$$

The third step is justified by the definition of stable wrt (U, X), which requires that $U \subset X$, and by the fact that $pos(r) \subset body(r)$. By Proposition 10.1, the consistent answer sets for $c_X(P)$ are the consistent answer sets for P. So we've shown that B is a consistent answer set for $b_U(P)$ iff there is a consistent answer set A for P s.t. $A \cap U = B$. □

Definition. Let x be a variable and t be a ground term in $\mathcal{L}_1$. Then $same\text{-}sort_{\mathcal{L}_2}(x, t)$ if x is a variable in $\mathcal{L}_2$, t is a ground term in $\mathcal{L}_2$ and t has the same sort in $\mathcal{L}_2$ as x does.

Definition. Let $[\neg]Q(t_1, \ldots, t_n) \in Lit(\mathcal{L}_1)$. For all i $(1 \leq i \leq n)$, $well\text{-}sorted_{\mathcal{L}_2}([\neg]Q(t_1, \ldots, t_n), i)$ if there is a literal $[\neg]Q(t'_1, \ldots, t'_n) \in Lit(\mathcal{L}_2)$ s.t. $t_i = t'_i$.

Intuitively, $well\text{-}sorted_{\mathcal{L}_2}([\neg]Q(t_1, \ldots, t_n), i)$ holds if Q is an n-ary predicate symbol of $\mathcal{L}_2$ and t_i is a term of $\mathcal{L}_2$ of the proper sort for the ith argument of Q in $\mathcal{L}_2$.

Definition. Let P be a program with I/O specification σ. Let $\mathcal{L}_1, \mathcal{L}_2$ be permissible languages for P.

$$
G^+_\sigma(\mathcal{H}(P, \mathcal{L}_1), \mathcal{L}_2) = \left\{ \begin{array}{c} [\neg]Q(t_1, \ldots, t_n) \in Lit(\mathcal{H}(P, \mathcal{L}_1)) : \\ for\ all\ i\ (1 \leq i \leq n),\ if\ \sigma_{[\neg]Q}(i) = +, \\ then\ well\text{-}sorted_{\mathcal{L}_2}([\neg]Q(t_1, \ldots, t_n), i) \end{array} \right\}
$$

Intuitively, $G^+_\sigma(\mathcal{H}(P, \mathcal{L}_1), \mathcal{L}_2)$ is the set of literals in $\mathcal{H}(P, \mathcal{L}_1)$ that are well-sorted in $\mathcal{L}_2$ in all argument places that are assigned $+$ by σ.

Lemma 10.6 *Let P be a program that is stable wrt I/O specification σ, with permissible languages $\mathcal{L}_1, \mathcal{L}_2$. Let $U = Lit(\mathcal{L}_1) \cap Lit(\mathcal{L}_2)$. Let $X = U \cup (Lit(\mathcal{L}_1) \setminus G^+_\sigma(\mathcal{H}(P, \mathcal{L}_1), \mathcal{L}_2))$. Program $\mathcal{H}(P, \mathcal{L}_1)$ is stable wrt (U, X).*

Proof. Assume $r \in \mathcal{H}(P, \mathcal{L}_1)$ s.t. $head(r) \cup body(r) \not\subset U$ and $head(r) \not\subset X \setminus U$. We must show that $pos(r) \not\subset X$. There is a rule $R \in P$, which is stable wrt σ, and for some ground substitution θ in $\mathcal{L}_1$, $r = R\theta$. Since $head(r) \cup body(r) \subset Lit(\mathcal{L}_1)$ but $head(r) \cup body(r) \not\subset U$, we conclude that there is a variable z in R s.t. $\neg same\text{-}sort_{\mathcal{L}_2}(z, z\theta)$. Since $head(R\theta) \not\subset X \setminus U$, there is a literal $L \in head(R)$ s.t. $L\theta \notin X \setminus U$. Thus, $L\theta \notin X$ or $L\theta \in U$. If $L\theta \notin X$, then $L\theta \in G^+_\sigma(\mathcal{H}(P, \mathcal{L}_1), \mathcal{L}_2)$. If $L\theta \in U = Lit(\mathcal{L}_1) \cap Lit(\mathcal{L}_2)$, then clearly, by the definition of G^+_σ, we again have $L\theta \in G^+_\sigma(\mathcal{H}(P, \mathcal{L}_1), \mathcal{L}_2)$. We can conclude that $z \notin FV^+(L)$, since $\neg same\text{-}sort_{\mathcal{L}_2}(z, z\theta)$. By the definition of stable wrt σ, there is an ordering $L_1, \ldots, L_k$ of the literals of $pos(R)$ s.t.

$$
\begin{array}{l}
[head(R) \neq \emptyset \wedge \forall b \in head(R)[z \in FV^+(b)]] \\
\quad \vee \\
\exists i \in \{1, \ldots, k\}[z \in FV^-(L_i) \wedge \forall j \in \{1, \ldots, i\}[z \notin FV^+(L_j)]] \, .
\end{array}
$$

Since we have shown that $z \notin FV^+(L)$ and $L \in head(R)$, it follows that

$$\exists i \in \{1,\ldots,k\}[z \in FV^-(L_i) \wedge \forall j \in \{1,\ldots,i\}[z \notin FV^+(L_j)]] .$$

Thus, we have shown that z occurs in some literal in $pos(R)$ and also that $\neg same\text{-}sort_{\mathcal{L}_2}(z, z\theta)$. Now, let L_i be the leftmost literal from $L_1,\ldots,L_k$ with a $y \in FV(L_i)$ s.t. $\neg same\text{-}sort_{\mathcal{L}_2}(y, y\theta)$. Clearly $L_i\theta \notin Lit(\mathcal{L}_2)$, so $L_i\theta \notin U$. It remains to show that $L_i\theta \in G^+_\sigma(\mathcal{H}(P,\mathcal{L}_1),\mathcal{L}_2)$. Since L_i is a literal in $\mathcal{L}_2$, if, for every $x \in FV^+(L_i)$, $same\text{-}sort_{\mathcal{L}_2}(x, x\theta)$, then $L_i\theta \in G^+_\sigma(\mathcal{H}(P,\mathcal{L}_1),\mathcal{L}_2)$, and we are done. So suppose there is an $x \in FV^+(L_i)$ s.t. $\neg same\text{-}sort_{\mathcal{L}_2}(x, x\theta)$. By the definition of stable, we can conclude that $\exists i' \in \{1,\ldots,i-1\}[x \in FV^-(L_{i'})]$, which contradicts our choice of L_i, since L_i is the leftmost literal from $L_1,\ldots,L_k$ with a $y \in FV(L_i)$ s.t. $\neg same\text{-}sort_{\mathcal{L}_2}(y, y\theta)$. So $L_i\theta \in G^+_\sigma(\mathcal{H}(P,\mathcal{L}_1),\mathcal{L}_2)$. We have shown that $L_i\theta \notin U$ and also that $L_i\theta \notin Lit(\mathcal{L}_1) \setminus G^+_\sigma(\mathcal{H}(P,\mathcal{L}_1),\mathcal{L}_2)$. That is, $L_i\theta \notin X$. Since $L_i\theta \in pos(r)$, we've shown that $pos(r) \not\subset X$, which was our goal. □

Lemma 10.7 *Let P be a stable program such that, for every permissible language $\mathcal{L}$ for P, every part of $\mathcal{H}(P,\mathcal{L})$ has a consistent answer set. Let $\mathcal{L}_1, \mathcal{L}_2$ be permissible languages for P, and let $U = Lit(\mathcal{L}_1) \cap Lit(\mathcal{L}_2)$. Then B is a consistent answer set for $b_U(\mathcal{H}(P,\mathcal{L}_1))$ iff there is a consistent answer set A for $\mathcal{H}(P,\mathcal{L}_1)$ s.t. $A \cap U = B$.*

Proof. Suppose P is stable wrt the I/O specification σ. Let $X = U \cup (Lit(\mathcal{L}_1) \setminus G^+_\sigma(\mathcal{H}(P,\mathcal{L}_1),\mathcal{L}_2))$. By Lemma 10.6, $\mathcal{H}(P,\mathcal{L}_1)$ is stable wrt (U,X). Since for every permissible language $\mathcal{L}$ for P, every part of $\mathcal{H}(P,\mathcal{L})$ has a consistent answer set, every part of $\mathcal{H}(P,\mathcal{L}_1)$ has a consistent answer set. Since $U = Lit(\mathcal{L}_1) \cap Lit(\mathcal{L}_2)$, it is clear that every literal in U has its complement in U. So, by Proposition 10.2, B is a consistent answer set for $b_U(\mathcal{H}(P,\mathcal{L}_1))$ iff there is a consistent answer set A for $\mathcal{H}(P,\mathcal{L}_1)$ s.t. $A \cap U = B$. □

Proof. (of Theorem 6.1) Suppose P is stable and for every permissible language $\mathcal{L}$ for P every part of $\mathcal{H}(P,\mathcal{L})$ has a consistent answer set. Let $\mathcal{L}_1, \mathcal{L}_2$ be permissible languages for P. Let $U = Lit(\mathcal{L}_1) \cap Lit(\mathcal{L}_2)$. Then $b_U(\mathcal{H}(P,\mathcal{L}_1)) = b_U(\mathcal{H}(P,\mathcal{L}_2))$. Suppose A_1 is a consistent answer set for $\mathcal{H}(P,\mathcal{L}_1)$. By Lemma 10.7, $A_1 \cap U$ is a consistent answer set for $b_U(\mathcal{H}(P,\mathcal{L}_1))$. Since $b_U(\mathcal{H}(P,\mathcal{L}_1)) = b_U(\mathcal{H}(P,\mathcal{L}_2))$, $A_1 \cap U$ is a consistent answer set for $b_U(\mathcal{H}(P,\mathcal{L}_2))$. By Lemma 10.7, there is a consistent answer set A_2 for $\mathcal{H}(P,\mathcal{L}_2)$ s.t. $A_2 \cap U = A_1 \cap U$. Since $A_1 \subset Lit(\mathcal{L}_1)$ and $A_2 \subset Lit(\mathcal{L}_2)$, it follows that $A_1 \cap Lit(\mathcal{L}_2) = A_2 \cap Lit(\mathcal{L}_1)$. So P is language tolerant. □

The following straightforward lemmas are used in the proof of Proposition 6.1 and are given here without proof.

Lemma 10.8 *If P is a predicate-order-consistent normal program, then for every permissible language $\mathcal{L}$ for P, $\mathcal{H}(P,\mathcal{L})$ is an order consistent normal program.*

Lemma 10.9 *Every part of an order-consistent normal program is also an order-consistent normal program.*

Proposition 10.3 *([Fages, 1993]) Every order-consistent normal program has an answer set.*

Proof. (of Proposition 6.1) Suppose P is a predicate-order-consistent normal program. By Lemma 10.8, for every permissible language $\mathcal{L}$ for P, $\mathcal{H}(P, \mathcal{L})$ is an order-consistent normal program. By Lemma 10.9, every part of $\mathcal{H}(P, \mathcal{L})$ is also an order-consistent normal program. So, by Proposition 10.3, for every permissible language $\mathcal{L}$, every part of $\mathcal{H}(P, \mathcal{L})$ has an answer set. Since normal programs have only consistent answer sets, the proposition is proved. □

Proof. (of Theorem 6.2) The theorem follows immediately by Proposition 6.1 and Theorem 6.1. □

Proof. (of Proposition 7.2) Let P be a language tolerant program, and let $\mathcal{L}$ be the language that is obtained from $\mathcal{L}_P$ by ignoring sorts. For the left-to-right direction, suppose that A is a consistent answer set for $\mathcal{H}(P, \mathcal{L}_P)$. By the definition of language tolerance, there is a consistent answer set A' for $\mathcal{H}(P, \mathcal{L})$ s.t. $A \cap Lit(\mathcal{L}) = A' \cap Lit(\mathcal{L}_P)$. Since $A \subset Lit(\mathcal{L}_P) \subset Lit(\mathcal{L})$, $A \cap Lit(\mathcal{L}) = A$. So $A = A' \cap Lit(\mathcal{L}_P)$. For the right-to-left direction, suppose there is a consistent answer set A' for $\mathcal{H}(P, \mathcal{L})$, and let $A = A' \cap Lit(\mathcal{L}_P)$. Since P is language tolerant, there is a consistent answer set A'' for $\mathcal{H}(P, \mathcal{L}_P)$ s.t. $A' \cap Lit(\mathcal{L}_P) = A'' \cap Lit(\mathcal{L})$. Since $A'' \subset Lit(\mathcal{L}_P) \subset Lit(\mathcal{L})$, $A' \cap Lit(\mathcal{L}_P) = A''$. So, $A = A''$. Thus, A is a consistent answer set for $\mathcal{H}(P, \mathcal{L}_P)$. □

Acknowledgements

The authors are grateful to Vladimir Lifschitz for useful discussions on the subject of this paper. We are also grateful to Enrico Giunchiglia and G. N. Kartha for their comments. This work was partially supported by the National Science Foundation under grants IRI-9101078 and IRI-9306751.

References

[Fages, 1993] François Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of logic in computer science*, 1993. To appear.

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

[Hanks and McDermott, 1987] Steve Hanks and Drew McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

[Lifschitz and Turner, 1994] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In *Logic Programming: Proceedings of the Eleventh International Conference for Logic Programming*, 1994.

[Lifschitz *et al.*, 1993] Vladimir Lifschitz, Norman McCain, and Hudson Turner. Automated reasoning about actions: a logic programming approach. Manuscript, 1993.

[Lloyd and Topor, 1986] John Lloyd and Rodney Topor. A basis for deductive database systems II. *Journal of Logic Programming*, 1:55–67, 1986.

[Martens and De Schreye, 1992] Bern Martens and Danny De Schreye. A perfect herbrand semantics for untyped vanilla meta-programming. Technical Report CW149, Department Computerwetenschappen, K.U. Leuven, Belgium, 1992.

[Ross, 1993] Kenneth A. Ross. On negation in HiLog. *Journal of Logic Programming*, 1993. To appear.

[Stroetman, 1993] Karl Stroetman. A completeness result for SLDNF resolution. *Journal of Logic Programming*, 15(4):337–355, 1993.

[Topor and Sonenberg, 1988] Rodney Topor and E.A. Sonenberg. On domain independent databases. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 217–240. Morgan Kaufmann, San Mateo, CA, 1988.