TLP **4** (3): 325–354, 2004. © 2004 Cambridge University Press
DOI: 10.1017/S1471068403001881 Printed in the United Kingdom

Applications of intuitionistic logic in Answer Set Programming

MAURICIO OSORIO, JUAN A. NAVARRO AND JOSÉ ARRAZOLA
Universidad de las Américas, CENTIA, Sta. Catarina Mártir, Cholula, Puebla, 72820 México
(*e-mail:* josorio@mail.udlap.mx)

Abstract

We present some applications of intermediate logics in the field of Answer Set Programming (ASP). A brief, but comprehensive introduction to the answer set semantics, intuitionistic and other intermediate logics is given. Some equivalence notions and their applications are discussed. Some results on intermediate logics are shown, and applied later to prove properties of answer sets. A characterization of answer sets for logic programs with nested expressions is provided in terms of intuitionistic provability, generalizing a recent result given by Pearce. It is known that the answer set semantics for logic programs with nested expressions may select non-minimal models. Minimal models can be very important in some applications, therefore we studied them; in particular we obtain a characterization, in terms of intuitionistic logic, of answer sets which are also minimal models. We show that the logic G_3 characterizes the notion of strong equivalence between programs under the semantic induced by these models. Finally we discuss possible applications and consequences of our results. They clearly state interesting links between ASP and intermediate logics, which might bring research in these two areas together.

KEYWORDS: answer sets, intuitionistic logic, equivalence, program transformations

1 Introduction

Answer Set Programming (ASP), Stable Logic Programming or A-Prolog, is the realization of much theoretical work on Non-monotonic Reasoning and AI applications of Logic Programming (LP) in the last 15 years. The main syntactic restriction needed in this paradigm is to eliminate function symbols from the language. This is because using infinite domains the answer sets are no longer necessarily recursively enumerable (Marek and Remmel, 2001). The two most well known systems that compute answer sets are `dlv`[1] and `smodels`[2].

Our work is intended to provide an alternative view of the theory of answer set programming through different tools and relations with intuitionistic and other intermediate logics. We provide a characterization of answer sets by intuitionistic

[1] `http://www.dbai.tuwien.ac.at/proj/dlv`
[2] `http://saturn.hut.fi/pub/smodels`

logic as follows:

"A formula is entailed by a logic program in the answer set semantics if and only if it can be proved in every intuitionistically complete and consistent extension of the program formed by adding only negated literals."

This is a generalization of a recent result given by Pearce where he considered disjunctive programs only. In our approach we consider the class of augmented programs, which allow nested formulas in the head and the body of clauses. Erdem and Lifschitz (2001) provided some evidence on how augmented programs can be used to represent and solve real life problems.

Our result provides foundations of defining the notion of non-monotonic inference of any propositional theory (using the standard connectives $\{\neg, \wedge, \vee, \rightarrow\}$) in terms of a monotonic logic (namely intuitionistic logic). We propose the following interpretation: We understand the *knowledge*, of a given theory T, as all the formulas F such that F is derived from T using intuitionistic logic. This makes sense since in intuitionistic logic, according to Brouwer (1981), A can be interpreted as "I know A". We will also identify a set of *beliefs* for the theory T. We will say it is safe to believe a formula F if and only if F belongs to every intuitionistically complete and consistent extension of T by adding only negated literals.

Take, for instance, $\neg a \rightarrow b$. The agent knows $\neg a \rightarrow b$, $\neg b \rightarrow \neg\neg a$ and so on. The agent, however, does not know neither a nor b. Nevertheless, one believes more than one knows. But a cautious agent must have his/her beliefs consistent to his/her knowledge. This agent will try to assume negated literals in order to infer more information. Thus, in our example, our agent can believe $\neg a$, since this assumption is consistent, in order to conclude b. At this point the agent can decide, for any formula constructed from a and b, either if it is true or false. The theory is now complete.

The agent could also try to assume $\neg b$ in order to conclude $\neg\neg a$, but he/she would not be able to intuitionistically prove a and the theory can not be completed. Thus it was not safe to believe $\neg b$. It also makes sense that a cautious agent could try to believe $\neg\neg a$ rather than to believe a (recall that a is not equivalent to $\neg\neg a$ in intuitionistic logic). Our results agree with the position of Kowalski (2001), namely "that Logic and LP need to be put into place: Logic within the thinking component of the observation-thought-action cycle of a single agent, and LP within the belief component of thought".

One important issue to know is when two programs are "equivalent" with respect to the answer set semantics. We consider a definition for "equivalence" that is given in Lifschitz *et al.* (2001). We say that P_1 and P_2 are strongly equivalent if for every program P, $P_1 \cup P$ and $P_2 \cup P$ have the same answer sets. If two programs are strongly equivalent, we know that we can replace one by the other in any larger program without changing the declarative semantics. This is an important concept for software engineering. It has been shown that the logic of Here-and-There (HT) or G_3 characterizes the class of strongly equivalent augmented programs under this definition (Lifschitz *et al.*, 2001).

If we want to use a program P_2 instead of another one P_1, it will be perfect if both programs have the same answer sets. This condition is, however, sometimes too much to expect. It will suffice if we can identify, through a simple relation, the answer sets of the first program knowing those of the second. A conservative extension (Osorio *et al.*, 2001) is one form of this weaker type of equivalence.

In order to define a "strong" version of this notion of equivalence we find useful to split the signature of programs into some user atoms and reserved atoms. The idea is that users are allowed to write programs using only the user atoms, while reserved atoms are used for internal program transformations. Given a user program P_1 and an internal program P_2, we say that P_2 is a strong conservative extension of P_1 if for every user program P, it holds that $P_2 \cup P$ is a conservative extension of $P_1 \cup P$. We show then that for every augmented program P there is a disjunctive program P' such that P' is a strong conservative extension of P. We also illustrate how to compute such a program P'.

Minimal models are of general interest for several theoretical and practical reasons (Bell *et al.*, 1993; Gelfond *et al.*, 1989; Liberatore, 1999; Lobo and Subrahmanian, 1992; Minker and Perlis, 1985). We therefore devote a section to study them in the context of answer sets. We first provide a characterization, in terms of intuitionostic logic, of answer sets that are also minimal models. And we show that two programs are strongly equivalent, with respect to the induced semantic, if and only if they are equivalent in the 3-valued logic G_3.

In this paper, we restrict our attention to finite propositional theories; the semantics can be extended to theories with variables by grounding. Function symbols are, however, not allowed to ensure the ground program to be finite. This is a standard procedure in ASP. We assume that the reader has some basic background in logic and Answer Set Programming.

Our paper is structured as follows. In section 2 we present the general syntax of clauses and define several types of programs. We also provide the definition of answer sets for augmented logic programs as well as some background on logic. In section 3 we present our notions of equivalence and provide some useful transformations to simplify the structure of programs. In section 4 we present our main result, the characterization of answer sets in terms of intuitionistic logic. In section 5 we study the class of answer sets that are minimal models. In section 6 we discuss several interesting consequences of the proposed approach and our main result. In section 7, we present some conclusions and ideas for future work. Finally as an appendix, in section Appendix A, we present the proofs of our results.

2 Background

In this section we review some basic concepts and definitions that will be used along this paper. We introduce first the syntax of formulas and programs based on the language of propositional logic. We also describe some common classes of logic programs and give the definition of answer sets. Finally, we make some comments on intermediate logics that will be used in later sections to study the notions of answer sets and non monotonic reasoning.

2.1 Propositional logic

We use the language of propositional logic in order to describe rules within logic programs. Formally, we consider a language built from an alphabet consisting of *atoms*: $p_0, p_1, \ldots$; *connectives*: $\wedge, \vee, \leftarrow, \bot$; and *auxiliary symbols*: '(', ')', '.'.

Where $\wedge, \vee, \leftarrow$ are 2-place connectives and $\bot$ is a 0-place connective. Formulas are defined as usual. The formula $\top$ is introduced as an abbreviation of $\bot \leftarrow \bot$, $\neg F$ as an abbreviation of $\bot \leftarrow F$, and $F \leftrightarrow G$ as an abbreviation of $(G \leftarrow F) \wedge (F \leftarrow G)$. The formula $F \rightarrow G$ is another way of writing the formula $G \leftarrow F$, we use the second form because of tradition in the context of logic programming.

A signature $\mathscr{L}$ is a finite set of atoms. If F is a formula then the *signature* of F, denoted as $\mathscr{L}_F$, is the set of atoms that occur in F. A *literal* is either an atom a (a positive literal) or a negated atom $\neg a$ (a negative literal). A *theory* is just a set of formulas.

2.2 Logic programs

A *logic program* is a finite set of formulas. The syntax of formulas within logic programs has been usually restricted to clauses with very simple structure. A *clause* is, in general, a formula of the form $H \leftarrow B$ where H and B are known as the *head* and *body* of the clause respectively. Two particular cases of clauses are *facts*, of the form $H \leftarrow \top$, and *constraints*, $\bot \leftarrow B$. Facts and constraints are sometimes written as H and $\leftarrow B$ respectively.

We introduce several kinds of clauses commonly found in literature. A *free clause* is built from a disjunction of literals in the head and a conjunction of literals in the body. Such a clause has the form

$$h_1 \vee \cdots \vee h_n \leftarrow b_1 \wedge \cdots \wedge b_m.$$

where each h_i and b_j is a literal. Either the head or the body of a free clause could be empty to denote a constraint or a fact. A *general clause* is a free clause that does not allow negation in the head, all literals in the head of the clause should be positive atoms. Finally, a *disjunctive clause* is a general clause with a non-empty head, i.e. it is not a constraint.

A *nested formula* is a formula built from the connectives $\wedge$, $\vee$ and $\neg$ arbitrarily nested. An *augmented clause* is a less restricted form of clause where both H and B can be nested formulas. Note, however, that embedded implications are not allowed in augmented clauses. The formula $a \leftarrow (b \rightarrow c)$ is not, for instance, an augmented clause. The following are examples of clauses just defined

$a \vee b \leftarrow c \wedge d \wedge \neg e.$	disjunctive, general, free, augmented
$\bot \leftarrow p \wedge q.$	general, free, augmented (constraint)
$a \vee \neg b \leftarrow p \wedge \neg q.$	free, augmented
$a \vee \neg a.$	free, augmented (fact)
$\neg(p \wedge \neg q) \leftarrow a \vee (\neg b \wedge c).$	augmented

We also say that a logic program is free if it contains only free clauses. Similarly, disjunctive and augmented programs are introduced. We would also use the term *logic program* alone to denote a set of arbitrary propositional formulas with no restrictions at all.

2.3 Answer sets

We now present the definition of answer sets for augmented programs. This material is taken from Lifschitz *et al.* (1999) with minor modifications since they consider a broader syntax of formulas. They consider two kinds of negation: default and classical. Our negation $\neg$ corresponds to their default negation *not*. Classical negation is not considered since it is easy to simulate it using a proper renaming of atoms. They also include an if-then-else constructor, but it is only an abbreviation of another formula. Hence, it is fair to say that their programs extend our augmented programs only by allowing the use of "classical" negation.

Atoms, as well as the connectives $\bot$ and $\top$, are called *elementary formulas.* Formulas built from $\wedge$ and $\vee$ over elementary formulas are called *basic.* Similarly basic clauses and programs are constructed from basic formulas. The definition of answer sets is given first for basic programs, without default negation, and is extended later to the class of augmented programs (Lifschitz *et al.*, 1999).

Definition 2.1
(Lifschitz *et al.*, 1999) We define when a set of atoms X *satisfies* a basic formula F, denoted $X \models F$, recursively as follows:

for elementary F, $X \models F$ if $F \in X$ or $F = \top$.
$X \models F \wedge G$ if $X \models F$ and $X \models G$.
$X \models F \vee G$ if $X \models F$ or $X \models G$.

Note that the previous definition does not contain the case of implication, since the syntax of augmented formulas does not allow to embed them as a subformula. Only one implication is allowed in each clause, and this is taken into account in the next definition.

Definition 2.2
(Lifschitz *et al.*, 1999) Let P be a basic program. A set of atoms X is *closed* under P if, for every clause $H \leftarrow B \in P$, $X \models H$ whenever $X \models B$.

Definition 2.3
(Lifschitz *et al.*, 1999) Let X be a set of atoms and P be a basic program. X is an *answer set* of P if X is minimal among the sets of atoms closed under P.

Definition 2.4
(Lifschitz *et al.*, 1999) The *reduct* of an augmented formula or program, relative to a set of atoms X, is defined recursively as follows:

for elementary F, $F^X = F$.
$(F \wedge G)^X = F^X \wedge G^X$.
$(F \vee G)^X = F^X \vee G^X$.
$(\neg F)^X = \bot$ if $X \models F^X$ and $(\neg F)^X = \top$ otherwise.
$(H \leftarrow B)^X = H^X \leftarrow B^X$.
$P^X = \{(H \leftarrow B)^X \mid H \leftarrow B \in P\}$.

Observe that the reduct of an agumented program, obtained as in previous definition, is a basic program. Using this reduct operator we are able to extend the definition of answer sets to the class of augmented programs.

Definition 2.5 (*Answer Sets*)
(Lifschitz *et al.*, 1999) Let P be an augmented program and X be a set of atoms. X is an *answer set* of P if it is an answer set of the reduct P^X.

Example 2.6
Consider the following program P:

$a \leftarrow \neg\neg a.$
$\neg b \leftarrow c \vee b.$

If we take $X = \{a\}$ then the reduct is P^X:

$a \leftarrow \top.$
$\top \leftarrow c \vee b.$

Here it is easy to verify that $\{a\}$ is closed under this reduct and, since the empty set $\emptyset$ is not, it is the minimal set with this property. Then it follows that $\{a\}$ is an answer set of P. However note that the empty set $\emptyset$ is also an answer set of P, since it produces a different reduct and is closed under it.

2.4 Intermediate logics

The main goal of the research presented in this paper is to study the current definition of answer sets in terms of mathematical logic. We present an extremely simple, logical characterization of answer sets applicable to augmented programs, based on a well-known alternative to classical logic, namely intuitionistic logic. Several interesting consequences of our approach are discussed in more detail in section 6.

We briefly describe in the following lines multivalued and intuitionistic logics. Interesting relations between these logics and the answer set semantics are studied in later sections. Some notation, definitions and simple results are given at the end of this section.

2.4.1 Gödel multivalued logics

These logics are defined generalizing the idea of truth tables and evaluation functions of classical logic. Gödel defined the multivalued logics G_i, with values in $\{0,1,\ldots,i-1\}$, with the following evaluation function I:

- $I(B \leftarrow A) = i-1$ if $I(A) \leqslant I(B)$ and $I(B)$ otherwise.
- $I(A \vee B) = \max(I(A), I(B))$.
- $I(A \wedge B) = \min(I(A), I(B))$.
- $I(\bot) = 0$.

An interpretation is a function $I : \mathscr{L} \to \{0,1,\ldots,i-1\}$ that assigns a truth value to each atom in the language. The interpretation of an arbitrary formula is obtained by propagating the evaluation of each connective as defined above. Recall that $\neg$ and $\top$ were introduced as abbreviations of other connectives. An interpretation is said to be *definite* if it assigns only values 0 or $i-1$, and *indefinite* if some intermediate value is assigned to an atom.

For a given interpretation I and a formula F we say that I is a *model* of F if $I(F)=i-1$. Similarly I is a *model* of a program P if it is a model of each formula contained in P. If F is modeled by every possible interpretation we say that F is a *tautology*. Notice that G_2 coincides with classical logic C. The 3-valued logic G_3 is particularly useful for some of our results.

2.4.2 *Intuitionistic logic*

This is an important logic, which has been an area of great interest during the last years. It is based on the concept of *proof* or *knowledge*, rather than *truth* in classical logic, to explain the meaning and use of logical connectives.

Intuitionistic logic, denoted I, can be defined in terms of Hilbert type proof systems of axioms and inference rules. Equivalent definitions can be given in terms of natural deduction systems and Kripke models (Troelstra and van Dalen, 1988; van Dalen, 1980). Surprisingly, no definition using a truth table scheme is possible. Provable formulas are called *theorems*. Gödel observed that there are infinitely many logics located between intuitionistic and classical logic (Zakharyaschev *et al.*, 2001). In particular, it has been shown that

$$\mathrm{I} \subset \cdots \subset \mathrm{G}_{i+1} \subset \mathrm{G}_i \subset \cdots \subset \mathrm{G}_3 \subset \mathrm{G}_2 = \mathrm{C}$$

where $\subset$ denotes proper inclusion of the set of provable formulas on each logic. We use the term *intermediate logic* to denote all logics, sets of classical tautologies closed under modus ponens and propositional substitution, that contains all the intuitionistic theorems. We say that a logic is a *proper intermediate logic* if it is an intermediate logic and is not the classical one. Observe that the multivalued logics G_i are intermediate logics.

2.4.3 *Notation and general definitions*

We use the standard notation $\vdash_{\mathrm{X}} F$ to denote that F is provable (a tautology, a theorem) in logic X. If T is a theory we understand the symbol $T \vdash_{\mathrm{X}} F$ to mean that $\vdash_{\mathrm{X}} F \leftarrow (F_1 \wedge \cdots \wedge F_n)$ for some formulas F_i contained in T. This is not the usual definition given in literature, but can be shown to be equivalent because of results like the *Deduction Theorem.* Similarly, if U is a theory, we use the symbol $T \vdash_{\mathrm{X}} U$ to denote $T \vdash_{\mathrm{X}} F$ for every $F \in U$.

A theory T is said to be consistent, with respect to logic X, if it is not the case that $T \vdash_{\mathrm{X}} \perp$. Also, a theory T is said to be (literal) *complete* if, for every atom $a \in \mathscr{L}_T$, we have either $T \vdash_{\mathrm{X}} a$ or $T \vdash_{\mathrm{X}} \neg a$. We say that a program is *incomplete* if it is not complete.

We use the notation $T \Vdash_{\mathrm{X}} U$ to stand for the phrase: T is consistent and $T \vdash_{\mathrm{X}} U$. Finally we say that two theories T_1 and T_2 are equivalent under logic X, denoted by $T_1 \equiv_{\mathrm{X}} T_2$, if it is the case that $T_1 \vdash_{\mathrm{X}} T_2$ and $T_2 \vdash_{\mathrm{X}} T_1$.

3 Equivalence notions

Given two programs we find useful to define several forms of equivalence relations. The most natural equivalence notion that can be defined in terms of the answer set semantics is that two programs are *equivalent* if they have exactly the same answer sets. However, this notion of equivalence is, sometimes too weak since it does not satisfy certain properties we would expect from an equivalence relation. Some other equivalence notions with richer properties need to be defined.

3.1 Strong Equivalence

Observe that, for instance, replacing equivalent pieces of programs in a larger program does not always ensure that the original and the transformed program are equivalent. The notion of *strong equivalence* is defined looking for this kind of properties.

Definition 3.1
(Lifschitz *et al.*, 2001) Two programs P_1 and P_2 are *strongly equivalent* if $P_1 \cup P$ is equivalent to $P_2 \cup P$ for every program P.

If two programs are strongly equivalent, we know that one of them can be replaced with the other in a larger program without changing the declarative semantics. It is clear that strong equivalence implies equivalence, but the converse is not true.

Example 3.2
Consider the programs $P_1 = \{a \leftarrow \neg b\}$ and $P_2 = \{a\}$, they are equivalent because $\{a\}$ is the unique answer set for both programs. However $P_1 \cup \{b \leftarrow a\}$ has no answer sets, while $P_2 \cup \{b \leftarrow a\}$ has the answer set $\{a, b\}$.

As a result of the study of strong equivalence of logic programs, an important relation between the answer set semantics and intermediate logics appeared in the following theorems.

Theorem 3.1
(Lifschitz *et al.*, 2001) Let P_1 and P_2 be two augmented programs. Then P_1 and P_2 are strongly equivalent iff P_1 and P_2 are equivalent in G_3 logic.

One intended use of this equivalence definition is to simplify programs. We can, for instance, translate an augmented program into a free program preserving strong equivalence.

Definition 3.3
Let P be an augmented program. Using distributive properties of conjunction, disjunction and negation (all of them valid in G_3 logic) rewrite each clause in P in the form $H \leftarrow B$, where H is a conjunction of simple disjunctions and B is a disjunction of simple conjunctions.

Using the following equivalences we can eliminate conjunctions in the head of clauses, disjunctions in the body and atoms with two (or more) negations:

$$A \wedge B \leftarrow C \quad \equiv_{G_3} \quad \begin{matrix} A \leftarrow C \\ B \leftarrow C \end{matrix} \qquad\qquad A \leftarrow B \vee C \quad \equiv_{G_3} \quad \begin{matrix} A \leftarrow B \\ A \leftarrow C \end{matrix}$$

$$A \vee \neg\neg B \leftarrow C \equiv_{G_3} A \leftarrow \neg B \wedge C \qquad\qquad A \leftarrow \neg\neg B \wedge C \equiv_{G_3} A \vee \neg B \leftarrow C$$

We write AugFree(P) to denote the resulting free program.

Example 3.4
We present now an example to explain how to compute the program AugFree(P) for a given program P. Suppose that we have the following augmented program P:

$\neg(a \wedge \neg b) \wedge c \leftarrow d \wedge (e \vee \neg f).$

We can introduce negations into subformulas, applying distributive properties of negation, until negation only appears in front of atoms:

$(\neg a \vee \neg\neg b) \wedge c \leftarrow d \wedge (e \vee \neg f).$

Now, using distributive properties of conjunction and disjunction, we can write the head (resp. body) of clauses in their normal conjunctive (resp. disjunctive) form:

$(\neg a \vee \neg\neg b) \wedge c \leftarrow (d \wedge e) \vee (d \wedge \neg f).$

The head consists now of a conjunction of disjunctions. Using one of the proposed equivalences we can remove all this conjunctions:

$\neg a \vee \neg\neg b \leftarrow (d \wedge e) \vee (d \wedge \neg f).$
$c \leftarrow (d \wedge e) \vee (d \wedge \neg f).$

Similarly, we proceed to remove disjunctions in the body:

$\neg a \vee \neg\neg b \leftarrow d \wedge e.$
$\neg a \vee \neg\neg b \leftarrow d \wedge \neg f.$
$c \leftarrow d \wedge e.$
$c \leftarrow d \wedge \neg f.$

We can finally remove atoms with two (or more) negations using the proposed equivalences:

$\neg a \leftarrow \neg b \wedge d \wedge e.$
$\neg a \leftarrow \neg b \wedge d \wedge \neg f.$
$c \leftarrow d \wedge e.$
$c \leftarrow d \wedge \neg f.$

This program obtained corresponds to what we call AugFree(P).

An immediate consequence, obtained by the construction of AugFree(P), is an equivalence relation with respect to the logic G_3.

Proposition 3.2
(Osorio *et al.*, 2001) Let P be an augmented program. Then P is equivalent under G_3 logic to the free program AugFree(P).

Using the machinery of logic we can conclude, from Theorem 3.1 and Proposition 3.2 above, that the defined transformation preserves strong equivalence. Formally we state the following theorem.

Theorem 3.3
(Lifschitz *et al.*, 1999) Let P be an augmented program. Then P is strongly equivalent to the free program AugFree(P).

Lifschitz *et al.* (1999) showed, using a similar transformation, that augmented programs can be translated into free programs without changing the corresponding answer sets. Just observe that Lifschitz *et al.* (1999) use the term "equivalence" to denote a "strong equivalence" as we introduced it here. We emphasize the fact that this result, with the language restricted to one kind of negation, can be obtained very easily through equivalence relations in logic.

Example 3.5
Consider the following augmented program P:

$a \leftarrow \neg\neg a.$
$\neg b \leftarrow c \vee b.$

It is possible to construct, applying the rules described in Definition 3.3, a free program which, by Theorem 3.3, is strongly equivalent to P. The program AugFree(P) obtained is:

$a \vee \neg a.$
$\neg b \leftarrow c.$
$\neg b \leftarrow b.$

3.2 Conservative extensions

If we want to use a program P_2 instead of another one P_1 it will be perfect if both programs have the same answer sets, but this condition is sometimes too much to expect. It will suffice, however, if we can identify through a simple relation the answer sets of the first program knowing those of the second. A *conservative extension* (Osorio *et al.*, 2001) is one form of this weaker type of equivalence.

Definition 3.6
Given two programs P_1 and P_2, we say that P_2 is a *conservative extension* of P_1 if it holds that M_1 is an answer set of P_1 iff M_2 is an answer set of P_2, where M_1 and M_2 satisfy $M_1 = M_2 \cap \mathscr{L}_{P_1}$.

Note that our definition is different from that in Baral (2003), since we do not ask for $P_1 \subseteq P_2$ to hold. In order to define a "strong" version of this equivalence notion we find it useful to split the signature of atoms, used to construct logic programs, into two disjoint sets $\mathscr{L}_U$ and $\mathscr{L}_R$ that we call the *user* and *reserved* signature respectively. Unless stated otherwise, we assume that logic programs are restricted to the user signature, such programs are called *user* programs. A program that is allowed to contain reserved atoms is called an *internal* program.

Definition 3.7
Given a user program P_1 and an internal program P_2, we say that P_2 is a *strong conservative extension* of P_1 if for every user program P, it holds that $P_2 \cup P$ is a conservative extension of $P_1 \cup P$.

The idea is that users are allowed to write programs using only atoms from the user signature. The reserved signature will be used when new atoms are needed to perform internal program transformations to, for instance, simplify the structure of programs and compute answer sets. The notion of strong conservative extension allows to apply such transformations locally to fragments of programs.

A well-known transformation, that preserves this kind of equivalence, has been used to translate general programs into disjunctive ones.

Definition 3.8
Given a general program $P = D \cup C$, written as a disjoint union where D is a disjunctive program and C the set of constraints in P. We define $\text{GenDis}(P) = D \cup \{p \leftarrow B \wedge \neg p \mid (\bot \leftarrow B) \in C\}$, where p is a new atom in $\mathscr{L}_R$.

The following lemma is a direct consequence of the behavior of this transformation, see Baral (2003).

Lemma 3.4
(Baral, 2003) Let P be a general program. $\text{GenDis}(P)$ is a strong conservative extension of P.

Sakama and Inoue (1998) showed that every free program can be transformed, through a conservative extension, into a general one. We use instead the more economical transformation presented in Osorio *et al.* (2001). Essentially, the same idea is presented in Definition 2 of Janhuenen (2001).

Definition 3.9
(Osorio *et al.*, 2001) Given a free program P, let S be the set containing all atoms a such that $\neg a$ appears in the head of some clause in P, and let φ be an injective function, $\varphi : S \rightarrow \mathscr{L}_R$, that assigns a new reserved atom to each element in S. Let P' be the program obtained from P by replacing each occurrence of $\neg a$ with $\varphi(a)$ for every atom $a \in S$, and let $\Delta_S = \bigcup_{a \in S} \{\varphi(a) \leftarrow \neg a, \bot \leftarrow a \wedge \varphi(a)\}$. Then we define $\text{FreeGen}(P) = P' \cup \Delta_S$.

Again, the following proposition is obtained as a direct consequence of results presented in Osorio *et al.* (2001).

Proposition 3.5
(Osorio *et al.*, 2001) Let P be a free program. $\text{FreeGen}(P)$ is a strong conservative extension of P.

Note that in this case, if we have already determined answer sets of P_2, it is possible to easily recover answer sets for P_1 just by taking the set intersection of each model with $\mathscr{L}_{P_1}$. It turns out that, in fact, if M is an answer set of P_1 then $M_S = M \cup \varphi(S \setminus M)$ is an answer set of P_2.

Example 3.10
Let P be the free program:

$$a \vee \neg a.$$

FreeGen(P) is the program:

$$a \vee x.$$
$$x \leftarrow \neg a.$$
$$\bot \leftarrow x \wedge a.$$

Recall that P has two answer sets $M_1 = \{\}$ and $M_2 = \{a\}$. We obtain, as expected, that FreeGen(P) has also two answer sets: $\{x\}$ and $\{a\}$.

Observe that if P_2 is obtained from P_1 by a finite sequence of strong conservative and/or strong equivalence transformations, then P_2 is also a strong conservative extension of P_1.

Using this transformations we can, starting from an augmented program P, construct $P_1 = \text{AugFree}(P)$, $P_2 = \text{FreeGen}(P_1)$ and $P_3 = \text{GenDis}(P_2)$. This chain of equivalences show that augmented programs are not more expressive than disjunctive ones under the answer set semantics. This means that, if we are able to compute answer sets of simple disjunctive programs, we can easily compute answer sets of more elaborated programs up to the augmented type. Formally we state the following theorem.

Theorem 3.6
For every augmented program P there is a disjunctive program P' such that P' is a strong conservative extension of P.

This result has also been presented in Pearce *et al.* (2002) where a polynomial transformation, based on a technique that involves renaming subformulas, is used instead of our AugFree(P). They even presented a working implementation[3] and proved nice properties like modularity (a consequence of the transformation being a strong conservative extension). For theoretical purposes any of these two transformations is equally valid in the following discussions.

4 Characterization of answer sets

In this section we present one of our main results in Theorem 4.2, which provides a characterization of answer sets of augmented programs in terms of intuitionistic logic and we propose a definition of answer sets for general propositional theories.

Given a signature $\mathscr{L}$ and a set of atoms $M \subseteq \mathscr{L}$ we define the complement of M as $\widetilde{M} = \mathscr{L} \setminus M$. The set $\mathscr{L}$ is not always given explicitly, we assume $\mathscr{L} = \mathscr{L}_P$ when a program P is clear by context.

Pearce (1999b) provided a first characterization of answer sets in terms of intuitionistic logic. He proved (his Theorem 3.4) that a formula is entailed by a *disjunctive* program in the answer set semantics if and only if it belongs to every

[3] `http://www.cs.uni-potsdam.de/~torsten/nlp/`

intuitionistically complete and consistent extension of the program formed by adding only negated atoms.

Theorem 4.1

(Pearce, 1999b) Let P be a disjunctive program. (i) If M is an answer set of P, then $P \cup \neg(\mathscr{L}_P \setminus M)$ is intuitionistically consistent and complete. (ii) Let $P \cup \Delta$ be intuitionistically consistent and complete, where $\Delta \subseteq \neg\mathscr{L}_P$; then $\{a \in \mathscr{L}_P \mid P \cup \Delta \vdash_{\mathrm{I}} a\}$ is an answer set of P.

Using our notation this theorem states that a set of atoms M is an answer set of P if and only if $P \cup \neg\widetilde{M} \Vdash_{\mathrm{I}} M$. This same result also holds for the class of general programs, we can allow the use of constraints. However, it fails to characterize answer sets if we allow negation in the head of clauses (free programs). Take for instance the free program $P = \{a \vee \neg a\}$. According to Definition 2.5 this program has two answer sets: $\{a\}$ and $\emptyset$. But only $\emptyset$, which corresponds to $\neg\widetilde{M} = \{\neg a\}$, satisfies Pearce's condition. For the other case, the condition is reduced to $a \vee \neg a \vdash_{\mathrm{I}} a$, but this is not even possible in classical logic.

We will see in the next section that the original approach from Pearce is actually characterizing another important notion in ASP, answer sets satisfying the condition in Theorem 4.1 are also minimal models.

However, to actually obtain the answer sets of a program, according to Definition 2.5, we propose to extend it not only with negated atoms, but allow twice negated atoms too. In the previous example, we would have $a \vee \neg a, \neg\neg a \vdash_{\mathrm{I}} a$, recovering the answer set $\{a\}$. We prove that this idea actually characterizes the notion of answer sets up to the class of augmented programs.

Theorem 4.2

Let P be an augmented program and M be a set of atoms. M is an answer set of P if and only if $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_{\mathrm{I}} M$.

This enhanced version of the theorem characterizes the notion of answer sets for augmented programs, and also provides a natural way to extend the definition of answer sets for logic programs containing arbitrary propositional formulas. Recall that the current definition of answer sets can only be applied to augmented programs, while the intuitionistic statement in Theorem 4.2 does not seem to imply any particular condition on the syntax of formulas in the program P. This will allow, for instance, the use of embedded implications inside clauses that were not allowed in augmented programs.

Here we sketch the idea of the proof followed by an example constructed over a particular instance. The main idea is to reduce augmented programs, using transformations described in the previous section, into disjunctive programs where we use Pearce's result as a starting point.

Suppose we have an *augmented* program P. We obtain first a *free* program $P_1 = \mathrm{AugFree}(P)$ by unwinding clauses in P. Now, negation in the head of clauses in P_1 can be eliminated to obtain a *general* program $P_2 = \mathrm{FreeGen}(P_1)$. Finally, constraints are removed to finish with a purely *disjunctive* program $P_3 = \mathrm{GenDis}(P_2)$.

As a consequence of equivalence theorems of the previous section, answer sets of our disjunctive program P_3 are related, by a simple one-to-one relation, with answer sets of P. We can apply the result from Pearce, Theorem 4.1, to the disjunctive program P_3 and traverse the chain of transformations backwards to recover the original program P.

First, we observe that answer sets of P_2 satisfy the same condition given by Pearce. This fact is obtained applying the following lemma to P_2.

Lemma 4.3
Let P be a general program and M be a set of atoms.
$\mathrm{GenDis}(P) \cup \neg(\mathscr{L}_{\mathrm{GenDis}(P)} \setminus M) \Vdash_{\mathrm{I}} M$ if and only if $P \cup \neg(\mathscr{L}_P \setminus M) \Vdash_{\mathrm{I}} M$.

Now, for the class of general programs, we can prove that both characterizations – proposed in Theorems 4.1 and 4.2 – coincide. Formally, we state the following lemma.

Lemma 4.4
Let P be a general program and M be a set of atoms.
$P \cup \neg\widetilde{M} \Vdash_{\mathrm{I}} M$ if and only if $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_{\mathrm{I}} M$.

The crucial step in the proof is the following lemma, it allows us to remove additional atoms added to the language of P_2 when using the transformation $\mathrm{FreeGen}(P_1)$. This would not be possible if we do not include the set $\neg\neg M$ to extend the program. The set S is obtained as in Definition 3.9, also recall that (by Proposition 3.5) the answer sets of P and $\mathrm{FreeGen}(P)$ are related by the identity $M_S = M \cup \varphi(S \setminus M)$.

Lemma 4.5
Let P be a free program and M be a set of atoms.
$\mathrm{FreeGen}(P) \cup \neg(\mathscr{L}_{\mathrm{FreeGen}(P)} \setminus M_S) \cup \neg\neg M_S \Vdash_{\mathrm{I}} M_S$ if and only if
$P \cup \neg(\mathscr{L}_P \setminus M) \cup \neg\neg M \Vdash_{\mathrm{I}} M$.

The final step is more simple, since the transformation of augmented to free programs already has some nice properties in terms of the G_3 logic. For our characterization we only need to show:

Lemma 4.6
Let P be an augmented program and M be a set of atoms.
$\mathrm{AugFree}(P) \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_{\mathrm{I}} M$ if and only if $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_{\mathrm{I}} M$.

Note that the language of $\mathrm{AugFree}(P)$ and P is the same. Following the chain of implications we are able to state that M is an answer set of the original P if and only if $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_{\mathrm{I}} M$. That is our Theorem 4.2. We clarify the idea of the proof with a concrete example.

Example 4.1
Consider again the augmented program P:

$a \leftarrow \neg\neg a.$
$\neg b \leftarrow c \vee b.$

As we know from Example 2.6 the set $M = \{a\}$ is an answer set for this program. Following Theorem 3.3, as done in Example 3.5, we construct the equivalent free program $P_1 = \text{AugFree}(P)$:

$$\begin{array}{l} a \vee \neg a. \\ \neg b \leftarrow c. \\ \neg b \leftarrow b. \end{array}$$

For this program we will replace atoms in $S = \{a, b\}$ that appear negated in the head of clauses with new atoms, as in Proposition 3.5, to build a general program, which is still equivalent. This program $P_2 = \text{FreeGen}(P_1)$ will contain

$$\begin{array}{ll} a \vee x. & x \leftarrow \neg a. \\ y \leftarrow c. & \bot \leftarrow a \wedge x. \\ y \leftarrow b. & y \leftarrow \neg b. \\ & \bot \leftarrow b \wedge y. \end{array}$$

with $M_S = \{a, y\}$ as the corresponding answer set. The final transformation $P_3 = \text{GenDis}(P_2)$ leads to the fully disjunctive program:

$$\begin{array}{ll} a \vee x. & x \leftarrow \neg a. \\ y \leftarrow c. & p \leftarrow a \wedge x \wedge \neg p. \\ y \leftarrow b. & y \leftarrow \neg b. \\ & p \leftarrow b \wedge y \wedge \neg p. \end{array}$$

Now we can apply Theorem 4.1 from Pearce and obtain a proof for the intuitionistic claim $P_3 \cup \{\neg b, \neg c, \neg x\} \Vdash_I \{a, y\}$. First, we can apply Lemma 4.3 to obtain $P_2 \cup \{\neg b, \neg c, \neg x\} \Vdash_I \{a, y\}$. Now, according to Lemma 4.4, we can include the facts $\neg\neg M_S$ in the intuitionistic formula $P_2 \cup \{\neg b, \neg c, \neg x, \neg\neg a, \neg\neg y\} \Vdash_I \{a, y\}$.

Recall that $P_2 = \text{FreeGen}(P_1) = P_1' \cup \Delta_S$ as written above. We can replace P_1' with P_1, as described in the proof of Lemma 4.5, to obtain the proof

$$P_1 \cup \{x \leftarrow \neg a, \bot \leftarrow a \wedge x, y \leftarrow \neg b, \bot \leftarrow b \wedge y, \neg b, \neg c, \neg x, \neg\neg a, \neg\neg y\} \Vdash_I a\,.$$

The atoms x and y added to the language of program when doing the transformation can now be mapped to the symbols $\bot$ and $\top$, respectively. We use this trick to eliminate them from the proof and to obtain:

$$P_1 \cup \{\bot \leftarrow \neg a, \bot \leftarrow a \wedge \bot, \top \leftarrow \neg b, \bot \leftarrow b \wedge \top, \neg b, \neg c, \neg\bot, \neg\neg a, \neg\neg\top\} \Vdash_I a\,.$$

This substitution works since, after some reductions, clauses originally contained in Δ_S are shown to be equivalent either to theorems or premises already listed. For this particular example formulas reduce to:

$$P_1 \cup \{\neg\neg a, \top, \top, \neg b\} \cup \{\neg b, \neg c, \top\} \cup \{\neg\neg a, \top\} \vdash_I a.$$

After removing such theorems, duplicate premises, and replacing P_1 with the original P, by Lemma 4.6, we finally obtain $P \cup \{\neg b, \neg c\} \cup \{\neg\neg a\} \Vdash_I a$.

The characterization provided by Theorem 4.2 has several important consequences. We have as an immediate result a characterization of equivalence of logic programs (under the answer set semantics) in terms of intuitionistic logic.

Corollary 4.7
Let P_1 and P_2 be two augmented programs sharing the same signature $\mathscr{L}$. P_1 and P_2 are equivalent if and only if, for every set of atoms $M \subseteq \mathscr{L}$, $P_1 \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_{\mathrm{I}} M$ iff $P_2 \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_{\mathrm{I}} M$.

Another nice feature is that this characterization allows us to generalize the notion of answer sets to programs containing arbitrary propositional formulas as clauses. We propose the following definition.

Definition 4.2
Let P be a logic program and M be a set of atoms. M is an *answer set* of P if $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_{\mathrm{I}} M$.

A similar extension for the notion of answer sets for arbitrary theories can be found in Lifschitz *et al.* (2001). They propose a generalization of answer sets in terms of the equilibrium logic introduced by Pearce (1999a). It is an interesting question left open to determine whether this two approaches are equivalent when dealing with arbitrary propositional theories. Our conjeture is that they, indeed, coincide.

In section 6, we will discuss in detail several benefits and consequences of such a definition. The important thing to observe is that this proposed definition provides a methodology of representing knowledge in a uniform way in the very well known intuitionistic logic, where known theoretical results can be applied to produce new interesting effects. Now, programs with implication in the body have a meaning (following Definition 4.2), and we can explore their use. Michael Gelfond points out (e-mail communication) that "the ability to use implication in the body seems to suggest the following translation:

r is true if every element with property p has property q. (*)
(Assume that the universe is finite)

The natural translation is

$$\forall X(p(X) \rightarrow q(X)) \rightarrow r.$$

If no implication is allowed in the formal language the translation of this English statement it loses its universal character. It now depends on the context and is prone to error." Hence, we believe that the use of our language could help to solve practical problems of representing knowledge. This makes sense since statements of type (*) are very frequent. In our language, a formula of the form $\forall X\alpha(X)$ could be introduced as an abbreviation of the conjunctive formula $\alpha(a_1) \wedge \ldots \wedge \alpha(a_n)$, where $\{a_1, \ldots, a_n\}$ is the Herbrand Universe of the program.

5 Answer sets and minimal models

In this section, we consider two valued interpretations, models and minimal models as usual in logic programming, see Lloyd (1987). Minimal models are of general interest for, at least, the following reasons: First, every answer set of a general program is a minimal model. Secondly, they are closely related to circumscription (Gelfond *et al.*, 1989; Minker and Perlis, 1985) and default logic (Lobo and Subrahmanian, 1992). Thirdly, they are of theoretical and practical interest for a large class of optimization problems (Liberatore, 1999). Finally, computation of minimal models can be the first step towards computing answer sets of general programs (see, for instance, Bell *et al.* (1993)).

In a few words, we can say that the set M is a minimal model of P if M is a model of P (with respect to classical logic), and it is minimal (with respect to set inclusion) among all other models of P. First, we provide a characterization of this notion in terms of provability in classical logic.

Lemma 5.1
For a given augmented program P, $P \cup \neg\widetilde{M} \Vdash_C M$ iff M is a minimal model of P.

We say that a set of atoms is a min-answer set if it is simultaneously a minimal model and an answer set. The following is also a main result of the paper. It provides a characterization of min-answer sets in terms of intuitionistic logic.

Theorem 5.2
Let P be an augmented program. M is a min-answer set of P iff $P \cup \neg\widetilde{M} \Vdash_I M$.

Observe that this is the same condition given by Pearce in Theorem 4.1. We conclude that he was actually characterizing min-answer sets and not answer sets in general. It turns out that answer sets of disjunctive programs are always minimal models, and thus both characterizations coincide on this restricted class of programs.

We also want to note that min-answer sets are not the same thing as minimal answer sets. By a minimal answer set we understand an answer set which is minimal among all answer sets of a program. The next proposition and the following example should make this difference clear.

Proposition 5.3
If M is a min-answer set of P, then M is a minimal answer set of P.

Example 5.1
The converse of Proposition 5.3 is not true. Let P the program:

$a \vee \neg a.$
$b \leftarrow a.$
$b \leftarrow \neg b.$

The unique answer set of P is $\{a, b\}$ and, since it is unique, it is also a minimal answer set of P. But $\{a, b\}$ is not a minimal model, since $\{b\}$ is the unique minimal model of P, hence P has no min-answer sets.

As a corollary of Theorems 3.6 and 5.2, we can conclude that the class of answer sets is not more expressive than the class of min-answer sets.

Corollary 5.4
For every augmented program P, there exists a computable disjunctive program P' such that the min-answer sets of P', restricted to $\mathscr{L}_P$, are each and every answer set of P.

The following theorem provides a characterization of strong equivalence, similar to the one in Theorem 3.1, for the class of min-answer sets. We observe that the logic G_3 can be used, again, to test for strong equivalence.

Theorem 5.5
Let P_1 and P_2 be two logic programs. Then P_1 and P_2 are strongly equivalent with respect to the min-answer set semantics if and only if P_1 and P_2 are equivalent in the logic G_3.

The proof is more complicated than that needed to prove Theorem 3.1 in Lifschitz *et al.* (2001), as we observe in Example 5.2. Moreover, we use the 3-valued logic G_3 instead of the Kripke semantics for HT.

Example 5.2
Consider the programs $P_1 = \{a \leftarrow a\}$ and $P_2 = \{a \vee \neg a\}$. These two programs are not strongly equivalent with respect to the answer set semantics simply because they are not equivalent. For the min-answer set semantics the situation is more complicated. Both programs are equivalent because $M = \emptyset$ is the unique min-answer set of each program. However, they are not strongly equivalent because, if we add the clause $P = \{a \leftarrow \neg a\}$ to each program, the two programs are no longer equivalent since $P_1 \cup P$ has no min-answer sets while $P_2 \cup P$ has exactly one min-answer set: $\{a\}$.

It is easy to verify, using the characterization of min-answer sets in Theorem 5.2, that programs equivalent under G_3 are strongly equivalent. For the converse we use the following two propositions that will allow us, under the assumption that two programs are not equivalent in G_3, to construct a third program that, when appended to the first two, will break equivalence with respect to the semantic of the min-answer sets.

Remark 5.6
Let A and B be two formulas. If $A \not\equiv_{\mathrm{G}_3} B$ then there is a 3-valued interpretation I which models A and not B (or models B and not A).

Proposition 5.7
Let P_1 and P_2 be arbitrary programs. If there is a 3-valued interpretation I such that I models P_1 and does not model P_2 then there exists a program P such that: (i) $P_1 \cup P$ is consistent and complete while $P_2 \cup P$ is inconsistent; or (ii) $P_1 \cup P$ is incomplete and cannot be completed (preserving consistency) by adding only negated atoms while $P_2 \cup P$ is both consistent and complete.

The following is an example to illustrate Proposition 5.7.

Example 5.3

Consider the programs $P_1 = \{a \leftarrow a\}$ and $P_2 = \{a \vee \neg a\}$. Let I be the 3-valued interpretation that evaluates $I(a) = 1$. Observe that I models P_1 (since $I(P_1) = 2$) and does not model P_2 (since $I(P_2) = 1$). Now if we take $P = \{a \leftarrow \neg a\}$ the program $P_1 \cup P$ becomes incomplete (and unable to be completed by adding negated atoms) while $P_2 \cup P$ is consistent and complete since it proves a. It turns out that $P_1 \cup P$ has no min-answer sets, while $P_2 \cup P$ has the min-answer set $\{a\}$.

We end this section with a question that we have not been able to answer yet. Suppose we have an augmented program P. Using Theorem 2 in Pearce *et al.* (2002), which states a transformation of augmented to disjunctive programs computable in polynomial time that preserves the answer set semantics, we can compute the corresponding disjunctive program P'.

Recall now that, in the restricted class of disjunctive programs, the semantics of answer sets and min-answer sets coincide. Thus, under the assumption that we have a min-answer set solver, we could compute the min-answer sets of P' which are exactly the answer sets of P'. By the properties of the transformation in Pearce *et al.* (2002) we can recover, by a very simple transformation, the answer sets of the original program P.

Our question is if there is a similar, polynomial time computable, transformation that could be used to compute min-answer sets for augmented programs under the assumption that we have an answer set solver. In other words, is the class of min-answer sets more expressive than the class of answer sets? If the answer is no, which is our conjecture, then both semantics would be equivalent in their power of representing problems. And there would be a great chance to obtain feedback between these two paradigms. On the other hand, if the answer is yes, then the min-answer sets would be more powerful than just answer sets. This would open a new line of research on the class of problems that could be expressed using the min-answer set semantics.

6 Applications and consequences

There are several nice consequences of Theorem 4.2. The first feature is that it provides a natural extension of the definition of answer sets for logic programs without depending on their particular restrictions of syntax or structure. It has been a usual approach to restrict the language of logic programs to some subsets of propositional logic while, the condition given in Theorem 4.2, does not imply any of such restrictions. We could use now, for instance, embedded implications in our programs, which are not allowed in the class of augmented formulas.

The proposed Definition 4.2 offers now an explanation of answer sets in terms of intuitionistic logic, where a wide variety of research has been done. We explore here some ideas that, thanks to results shown in this paper, allow us to better understand and generalize the notion of answer sets.

6.1 Safe beliefs

Consider a logic agent, whose base knowledge of the world and its behavior is described by a set of propositional formulas P. Under the premise that P is consistent, our agent can start infering from this base knowledge. Intuitionistic logic, a logic of knowledge, seems to be a natural inference system for this approach. We can say then that our agent *knows* F, a propositional formula in general, if $P \Vdash_I F$.

However, we also want our agent to be able to do non-monotonic inference. Informally speaking we allow our agent to *guess* or *suppose* things in order to make more inference. But there is no reason, however, to just believe everything that seems possible. We only *suppose* facts if there is some reason to believe them or, more precisely, if they are helpful to produce any new knowledge.

Under this context we can rephrase the definition of answer sets. For this we introduce the symbol $\overline{M}$, the *closure* of M, as $\overline{M} = M \cup \neg\widetilde{M}$. This $\overline{M}$ contains a complete set of beliefs for our agent, the following definition states when this set of beliefs can be considered as safe.

Definition 6.1

Let P be a logic program and M be a set of atoms. Then $\overline{M}$ is a set of *safe beliefs* if it satisfies $P \cup \neg\neg\overline{M} \Vdash_I \overline{M}$.

A very natural reading of the previous definition in the described context is: *"If a set of beliefs $\overline{M}$ is (i) consistent with the base knowledge and (ii) if we can suppose that the facts contained in $\overline{M}$ are true, and this is enough to be sure about this facts, then it is safe to believe $\overline{M}$."* This *suppose* corresponds to the double negation in the intuitionistic statement. Observe that safe beliefs, defined this way, exactly correspond to the answer sets of P.

An immediate benefit of such a definition is that it extends the syntax of programs allowing embedded implication in clauses. This broader syntax can allow us to write some rules for describing problems in a more natural way. We even suspect that this kind of syntax can be helpful to model concepts like aggregation in logic programs, as the ones described in Osorio and Jayaraman (1999) and Osorio *et al.* (1999). Further research has to be done in this direction to present more concrete results.

On the theoretical point of view, there are also several benefits provided by this approach. Equivalence notions can be easily described in terms of logic. The fact that logic G_3 characterizes strong equivalence can be proved to hold for answer sets under this new definition. This is done in Navarro (2002), where a proof, that does not depend on the syntax of formulas, is given.

Other interesting result, presented in Osorio *et al.* (2002c), is that the proposed definition of answer sets does not strictly depend on the underlying logic. It is proved that any proper intermediate logic does define the same semantic.

We try to demonstrate with this arguments various interesting possibilities on answer sets using logic. We show in Osorio *et al.* (2002b) how G_3 can be used to debug a progam by taking advantage of the 3-valued nature of G_3.

6.2 Answer sets in other logics

In this paper, we do not consider the so-called classical negation, but it can easily be included in the same intuitionistic framework by a simple renaming method (Baral, 2003; Gelfond and Lifschitz, 1990). However, if we are interested in using this classical negation with all of its power we can just replace intuitionistic logic by a Nelson logic in our proposed Definition 4.2 – see Osorio *et al.* (2002).

Another interesting extension we can provide, due to this intuitionistic characterization, is a definition of answer sets for logic programs containing modal formulas. Modal logics were originated when trying to formalize notions like *necessary* and *possible* in logic. The new pair of connectives $\mathscr{K}$ and $\mathscr{B}$ introduced have been also interpreted to model similar notions like tense, moral obligation and knowledge.

Using the well-known Gödel embedding of intuitionistic logic into modal logic S4 we can provide a natural definition of answer sets based on this logic. The actual definition of the Gödel mapping $^\circ$, that satisfies $\vdash_{\mathrm{I}} A$ if and only if $\vdash_{\mathrm{S4}} A^\circ$, can be found in Zakharyaschev *et al.* (2001).

We have to define some sort of *basic acceptable knowledge* as unary formulas, containing just one atom and unary connectives, that will play the role of the $\overline{M}$ above. For a modal logic program P and a *complete* set of basic acceptable knowledge M it would be reasonable to define M as an *answer set*, or *safe beliefs* so to say, of P if it satisfies $P \cup \mathscr{K}\mathscr{B}M \Vdash_{\mathrm{S4}} M$. More precise definitions have to be given, but the main idea should be clear.

The use of modal formulas seems very appropriate, since we would be able to explicitly model the concept of knowledge in logic programs. Then the generalization to multimodal logics should be a natural extension. This could provide a general framework where multiple agents can simultaneously reason, with the power of non-monotonic inference, about the knowledge and beliefs of each other. Some advances in this research line are presented in Osorio *et al.* (2002a).

A similar exercise can be done to define answer sets using linear logic. It is known that intuitionistic logic can be embedded in the propositional fragment of linear logic (Girard, 1987). Thanks to our results and the given embedding, it is possible to define the notion of an answer set in an environment with limited resources and thus provide some foundations for a framework of ASP as provability in linear logic – see Osorio *et al.* (2002).

7 Conclusions and related work

Work that relates ASP with classical logic can be found in Eshgi and Kowalski (1989) and Niemelä and Simons (1996). Erdem and Lifschitz relate answer sets and supported models in Erdem and Lifschitz (2001). Work that relates ASP with epistemic and modal logic can be found in Lifschitz and Schwarz (1993) and Marek and Truszczyński (1993). Our work follows the approach started by Pearce (1999b). We generalize his characterization, given for disjunctive programs, to augmented programs. We also study the class of answer sets which are also minimal models (min-answer sets) not done before to our knowledge.

We provide a characterization of answer sets in intuitionistic logic as follows: a formula is entailed by an augmented program in the answer set semantics if and only if it is proved in every intuitionistically complete and consistent extension of the program formed by adding only negated literals. As we explain in the introduction, our result provides the foundations to explain the notion of non-monotonic inference of any theory (using the standard connectives $\{\neg, \wedge, \vee, \leftarrow\}$) in terms of a monotonic logic (namely intuitionistic logic).

An immediate application of our result is to be able to have a definition of ASP for arbitrary propositional theories. A similar result was presented in Lifschitz *et al.* (2001) where a generalization of answer sets in terms of the, so called, equilibrium logic is stated. Our result provides, in particular, a natural way to extend the notion of answer sets in other logics.

Of particular interest to us are multimodal logics, because they can be used to naturally model the interaction of several agents. However, this is an open problem as we have explained. We believe that, in general, our results presented in this paper re-emphasize that the approach of answer sets is a solid paradigm to model non-monotonic reasoning.

We find a characterization of min-answer sets in terms of intuitionistic logic. We observe that, in some way, the class of answer sets is no more expressive than the class of min-answer sets. We may ask: "Is the class of min-answer sets more expressive than the class of answer sets?" We argue that, no matter what the answer is, it will have impact in the theory of ASP.

Our results are given for propositional theories but they can easily be generalized to universally quantified theories without functional symbols. It would be interesting, however, to generalize our results to arbitrary first order theories. Due to the large amount of knowledge in intuitionistic logic, we expect to obtain a lot of feedback between these two areas.

Acknowledgements

Numerous discussions with Michael Gelfond helped us to clarify our ideas. This research is sponsored by the Mexican National Council of Science and Technology, CONACyT (projects 35804-A and 37837-A).

Appendix A Proofs

In this section we present references to some basic results and the proofs of our main theorems and propositions in this paper.

A.1 Basic results

The following are some basic results and definitions that were proved in other sources and will be important for the proofs of our new results.

Lemma Appendix A.1
(van Dalen, 1980) Let T be any theory, and let F, G be a pair of equivalent formulas (under any intermediate X). Any theory obtained from T by replacing some occurrences of F by G is equivalent to T (under logic X).

Lemma Appendix A.2
(Osorio *et al.*, 2001) Let T_1, T_2 be two theories and let A be a formula such that $\mathscr{L}_{T_1 \cup \{A\}} \cap \mathscr{L}_{T_2} = \emptyset$. If T_2 is a set of negative literals and $T_1 \cup T_2 \vdash_I A$ then $T_1 \vdash_I A$.

Definition Appendix A.1
(Osorio *et al.*, 2001) The set $\mathbf{P}$ of *positive formulas* is the smallest set containing all formulas without negation connectives ($\neg$). The set $\mathbf{N}$ of *two-negated formulas* is the smallest set $\mathbf{X}$ with the properties:

1. If a is an atom then $(\neg\neg a) \in \mathbf{X}$.
2. If $A \in \mathbf{X}$ then $(\neg\neg A) \in \mathbf{X}$.
3. If $A, B \in \mathbf{X}$ then $(A \wedge B) \in \mathbf{X}$.
4. If $A \in \mathbf{X}$ and B is any formula then $(A \vee B), (B \vee A), (A \leftarrow B) \in \mathbf{X}$.

For a given set of formulas Γ, we define the *positive subset* of Γ, denoted $\mathrm{Pos}(\Gamma)$, as the set $\Gamma \cap \mathbf{P}$.

Proposition Appendix A.3
(Osorio *et al.*, 2001) Let Γ be a subset of $\mathbf{P} \cup \mathbf{N}$, and let $A \in \mathbf{P}$ be a positive formula. If $\Gamma \vdash_I A$ then $\mathrm{Pos}(\Gamma) \vdash_I A$.

A.2 Proofs about equivalence

Proof of Theorem 3.6
Let P be an augmented program. By Theorem 3.3, $P_1 = \mathrm{AugFree}(P)$ is strongly equivalent to P. Then, by Proposition 3.5, $P_2 = \mathrm{FreeGen}(P_1)$ is a conservative extension of P_1. Similarly, by Lemma 3.4, $P_3 = \mathrm{GenDis}(P_2)$ is a conservative extension of P_2. Through the chain of equivalences we obtain that P_3 is a conservative extension of P. □

A.3 Reductions for general programs

We present here some definitions and simple results of reductions, motivated by results in Dix *et al.* (2001), for the class of general programs. They are helpful in the proof of Theorem 4.2, particularly at Lemma 4.4.

Definition Appendix A.2 (*First Reduction*)
(Osorio *et al.*, 2002c) Let P be a general program and M be a set of atoms. We define the *first reduction* of P with respect to $\neg M$, denoted $\mathrm{Redu1}(P; \neg M)$, as the program obtained applying the following transformation to each clause $H \leftarrow B$ contained in P:

- Delete from B all literals $\neg a$ such that $\neg a \in \neg M$.
- Delete from H all literals a such that $\neg a \in \neg M$.
- Delete the clause if there is some literal a in B such that $\neg a \in \neg M$.

Lemma Appendix A.4
Let P be a general program and M be a set of atoms. The second reduction satisfies $P \cup \neg\neg M \equiv_{\mathrm{I}} \mathrm{Redu2}(P;\neg\neg M) \cup \neg\neg M$.

Definition Appendix A.3 (*Second Reduction*)
Let P be a general program and let M be a set of atoms. We define the *second reduction* of P with respect to $\neg\neg M$, denoted $\mathrm{Redu2}(P;\neg\neg M)$, as the program obtained applying the following transformation to each clause $H \leftarrow B$ contained in the program P:

- If $H = \bot$ then delete from B all literals a such that $\neg\neg a \in \neg\neg M$.
- Delete the clause if there is some literal $\neg a$ in B such that $\neg\neg a \in \neg\neg M$.

Lemma Appendix A.5
(Osorio *et al.*, 2002c) Let P be a general program and M be a set of atoms such that $P \cup \neg M$ is consistent. If $P' = \mathrm{Redu1}(P;\neg M)$ then the following properties hold:

1. $P \cup \neg M \equiv_{\mathrm{I}} P' \cup \neg M$.
2. $\mathscr{L}_{P'} \cap M = \emptyset$.

Proof
The two transformation steps in the reduction can be justified since it is possible to show, using intuitionistic logic, $\neg\neg a \vdash_{\mathrm{I}} (\bot \leftarrow a \wedge B) \leftrightarrow (\bot \leftarrow B)$ and $\neg\neg a \vdash_{\mathrm{I}} H \leftarrow \neg a \wedge B$, respectively. □

A.4 Proofs about the characterization of answer sets

Proof of Lemma 4.3
In the following paragraph the set $\widetilde{M}$ always represents the set $\mathscr{L}_P \setminus M$, that is the set complement of M with respect to the signature of the program P. Observe that, making this assumption, the set $\mathscr{L}_{\mathrm{GenDis}(P)} \setminus M = (\mathscr{L}_P \cup \{p\}) \setminus M = \widetilde{M} \cup \{p\}$.

The transformation step that defines GenDis(P) preserves equivalence in intuitionistic logic, since $\neg p \vdash_{\mathrm{I}} (\bot \leftarrow B) \leftrightarrow (p \leftarrow B \wedge \neg p)$. So in particular $\mathrm{GenDis}(P) \cup \neg\widetilde{M} \cup \{\neg p\} \Vdash_{\mathrm{I}} M$ iff $P \cup \neg\widetilde{M} \cup \{\neg p\} \Vdash_{\mathrm{I}} M$ iff, by Lemma Appendix A.2 and since $p \notin \mathscr{L}_P$, $P \cup \neg\widetilde{M} \Vdash_{\mathrm{I}} M$. □

Proof of Lemma 4.4
First suppose $P \cup \neg\widetilde{M} \Vdash_{\mathrm{I}} M$. Since $A \rightarrow \neg\neg A$ is an intuitionistic theorem $P \cup \neg\widetilde{M} \Vdash_{\mathrm{I}} \neg\neg M$. Therefore $P \cup \neg\widetilde{M} \cup \neg\neg M$ is consistent and since intuitionistic logic is monotone we have $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_{\mathrm{I}} M$.

Now suppose $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_{\mathrm{I}} M$, it is immediate that $P \cup \neg\widetilde{M}$ is consistent. Now, we want to show that $P \cup \neg\widetilde{M} \vdash_{\mathrm{I}} M$. If $P' = \mathrm{Redu1}(P;\neg\widetilde{M})$ then, by Lemma Appendix A.5, $P \cup \neg\widetilde{M} \equiv_{\mathrm{I}} P' \cup \neg\widetilde{M}$ and $\mathscr{L}_{P'} \cap \widetilde{M} = \emptyset$.

Since $P \cup \neg\widetilde{M} \cup \neg\neg M \vdash_{\mathrm{I}} M$, then $P' \cup \neg\widetilde{M} \cup \neg\neg M \vdash_{\mathrm{I}} M$. It is also clear that $\mathscr{L}_{P' \cup \neg\neg M} \cap \mathscr{L}_{\neg\widetilde{M}} = \emptyset$, so we can apply Lemma Appendix A.2 to get $P' \cup \neg\neg M \vdash_{\mathrm{I}} M$.

Let $P'' = \mathrm{Redu2}(P';\neg\neg M)$. Since $\mathscr{L}_{P'} \cap \widetilde{M} = \emptyset$, that $\mathscr{L}_{P'} \subseteq M$. So, by definition of Redu2, disjunctive clauses containing negative literals in P' will be always removed. Also, note that P'' can not contain constraints. If P' contains a constraint of the form $\bot \leftarrow B$ and is not removed is because only positive literals occur in B. However redu2 will, for this clause, remove all literals in the body and leave a clause $\bot \leftarrow \top$. But this is a contradiction, since we already know that $P \cup \neg\widetilde{M} \cup \neg\neg M$ is consistent. So P'' is an entirely positive program and, moreover, $P'' \subseteq P'$.

Since, by Lemma Appendix A.4, $P' \cup \neg\neg M \equiv_I P'' \cup \neg\neg M$ we have $P'' \cup \neg\neg M \vdash_I M$ and, using Proposition Appendix A.3, $P'' \vdash_I M$. But we already know that $P \cup \neg\widetilde{M} \vdash_I P' \cup \neg\widetilde{M}$ and, since $P'' \subseteq P'$, in particular $P \cup \neg\widetilde{M} \vdash_I P''$. So finally we obtain, as desired, $P \cup \neg\widetilde{M} \vdash_I M$. □

Proof of Lemma 4.5

Suppose $\text{FreeGen}(P) \cup \neg\widetilde{M_S} \cup \neg\neg M_S \Vdash_I M_S$. Recall that $\text{FreeGen}(P)$ can be written as $P' \cup \Delta_S$ and, since $P \cup \Delta_S \vdash_I P'$, $P \cup \Delta_S \cup \neg\widetilde{M_S} \cup \neg\neg M_S \Vdash_I M_S$. Since $M_S = M \cup \varphi(S \setminus M)$ we can break the sets $\neg\widetilde{M_S}$ and $\neg\neg M_S$ as disjoint subsets from user and reserved atoms. That is $\neg\widetilde{M_S} = \neg\widetilde{M} \cup \neg[\varphi(S \cap M)]$ and $\neg\neg M_S = \neg\neg M \cup \neg\neg[\varphi(S \setminus M)]$.[4]

Similarly we can write Δ_S as the disjoint union $\Delta_M \cup \Delta_{(S \setminus M)}$. Then, since $\neg\neg M \cup \neg[\varphi(S \cap M)] \vdash_I \Delta_M$, we have that

$$P \cup \Delta_{(S \setminus M)} \cup \neg\widetilde{M} \cup \neg\neg M \cup \neg[\varphi(S \cap M)] \cup \neg\neg[\varphi(S \setminus M)] \Vdash_I M .$$

In the intuitionistic proof for each $a \in M$ as shown above we can map each symbol $x \in \varphi(S \cap M)$ to $\bot$ and each symbol $y \in \varphi(S \setminus M)$ to $\top$. This will lead to a proof for $P \cup \neg\widetilde{M} \cup \neg\neg M \vdash_I M$, since premises in $\Delta_{(S \setminus M)}$, $\neg[\varphi(S \cap M)]$ and $\neg\neg[\varphi(S \setminus M)]$ are mapped either to intuitionistic theorems or elements in $\neg\widetilde{M}$.

To prove the other implication assume $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$. From definition of Δ_S we have that $P \cup \Delta_S \cup \neg\widetilde{M} \cup \neg\neg M$ is consistent, also note that $\Delta_S \cup \neg\widetilde{M} \cup \neg\neg M \vdash_I \varphi(S \setminus M) \cup \neg[\varphi(S \cap M)]$. It follows then that

$$P' \cup \Delta_S \cup \neg\widetilde{M} \cup \neg\neg M \cup \neg[\varphi(S \cap M)] \cup \neg\neg[\varphi(S \setminus M)] \Vdash_I M \cup \varphi(S \setminus M)$$

as we wanted. □

Proof of Lemma 4.6

By construction we have $P \equiv_{G_3} \text{AugFree}(P)$. In particular, for every set of atoms M, this implies $P \cup \neg\widetilde{M} \cup \neg\neg M \equiv_I \text{AugFree}(P) \cup \neg\widetilde{M} \cup \neg\neg M$. So we finally obtain $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$ iff $\text{AugFree}(P) \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$. □

Proof of Theorem 4.2

The set of atoms M is an answer set of the augmented program P iff, by Theorem 3.3, M is an answer set of $P_1 = \text{AugFree}(P)$ iff, by Proposition 3.5, M_S is an answer set of $P_2 = \text{FreeGen}(P_1)$ iff, by Lemma 3.4, M_S is an answer set of $P_3 = \text{GenDis}(P_2)$ iff, by Theorem 4.1, $P_3 \cup \neg\widetilde{M_S} \Vdash_I M_S$ iff, by Lemma 4.3, $P_2 \cup \neg\widetilde{M_S} \Vdash_I M_S$ iff, by Lemma 4.4, $P_2 \cup \neg\widetilde{M_S} \cup \neg\neg M_S \Vdash_I M_S$ iff, by Lemma 4.5, $P_1 \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$ iff, by Lemma 4.6, $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$. □

[4] Note that $\widetilde{M} = \mathscr{L}_P \setminus M$, while $\widetilde{M_S} = (\mathscr{L}_P \cup \varphi(S)) \setminus M_S$.

Proof of Corollary 4.7

We have that the two programs P_1 and P_2 are equivalent in the answer set semantics iff, by definition, (M is an answer set of P_1 iff M is an answer set of P_2) iff, by Theorem 4.2, ($P_1 \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$ iff $P_2 \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$). $\square$

A.5 Proofs about answer sets and minimal models

Proof of Lemma 5.1

Suppose that $P \cup \neg\widetilde{M} \Vdash_C M$. Then M is model of P. Suppose then that M is not a minimal model of P. Then there exists N, a model of P such that $N \subset M$, take $a \in M \setminus N$. In particular $\neg a \in \neg\widetilde{N}$, thus $P \cup \neg\widetilde{N} \vdash_I \neg a$. But, since $P \cup \neg\widetilde{M} \vdash_I a$ and $\neg\widetilde{M} \subseteq \neg\widetilde{N}$, $P \cup \neg\widetilde{N} \vdash_I a$ and N is not a model.

For the converse, if M is a minimal model of P then P is consistent (it has one model) and $P \cup \neg\widetilde{M}$ is also consistent. But it is easy to check that M, since it is minimal, is the unique model of $P \cup \neg\widetilde{M}$. So $P \cup \neg\widetilde{M} \vdash_C M$. $\square$

Proof of Theorem 5.2

Suppose that M is a min-answer set of P so $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$, because M is an answer set of P. Since M is a minimal model, we know that $P \cup \neg\widetilde{M} \vdash_C M$, by Lemma 5.1, then $P \cup \neg\widetilde{M} \vdash_I \neg\neg M$. By the last assertion and since $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_I M$, we have $P \cup \neg\widetilde{M} \vdash_I M$ and $P \cup \neg\widetilde{M}$ is consistent, i.e. $P \cup \neg\widetilde{M} \Vdash_I M$.

For the converse, suppose $P \cup \neg\widetilde{M} \Vdash_I M$ so $P \cup \neg\widetilde{M} \vdash_I \neg\neg M$, hence $P \cup \neg\widetilde{M} \cup \neg\neg M$ is consistent in intuitionistic logic and M is an answer set of P. On the other hand, if $P \cup \neg\widetilde{M} \Vdash_I M$ then $P \cup \neg\widetilde{M} \vdash_C M$ and we know that $P \cup \neg\widetilde{M}$ is consistent in intuitionistic logic, which implies consistency in classical logic. By Lemma 5.1, we have that M is a minimal model of P. $\square$

Proof of Proposition 5.3

Suppose that M is a min-answer set of P, but P is not a minimal answer set of P. Then there is $N \subset M$, such that N is a minimal answer set of P. In particular N is a model of P. But this is not correct, since M is a minimal model of P. $\square$

Proof of Corollary 5.4

Follows by Theorem 3.6 and the well known fact that, for disjunctive programs, the answer sets are minimal models. $\square$

Proof of Proposition 5.7

We assume that I' is defined as in the proposition above. Observe in particular that, if I models A then I' models A too.

Consider also the following definition: For a given interpretation I in G_3 the program $T(I)$ is defined as the minimum set X which satisfies

1. If $I(a)=I(b)=1$ and $a \neq b$ then $(b \leftarrow a) \in X$.
2. If $I(a)=1$ then $(a \leftarrow \neg a) \in X$.
3. If $I(a)=2$ then $(a) \in X$.
4. If $I(a)=0$ then $(\bot \leftarrow a) \in X$.

We have two main cases in the proof of this proposition:

1. There is a *definite* interpretation I that models P_1 and not P_2. Let $P = T(I)$ (as just defined). Then, by construction, $P_1 \cup P$ is a consistent and complete extension of P_1, while $P_2 \cup P$ is inconsistent.
2. If every interpretation that models P_1 and not P_2 is indefinite, then let I be one of such interpretations and $P = T(I)$. Since I models P_1 we have that I' models P_1 too. Notice that $I \neq I'$ since I contains some 1 assignments and I' not.

Since I models P, we have I models $P_1 \cup P$. Again I' models $P_1 \cup P$ too. But $I \neq I'$ so there are two different interpretations that model $P_1 \cup P$ and hence it is not complete. Nor it can be completed by adding negated atoms $\neg a$ since the program will become inconsistent (if $I(a)=1$ or $I(a)=2$) or still be incomplete (if $I(a)=0$).

We prove now that I' models P_2. If not I' will model P_1 and not P_2 but I' is definite contradicting the hypothesis of this case. Again, since I' models P, I' models $P_2 \cup P$.

It will be shown that I' is the only interpretation that models $P_2 \cup P$. Suppose K is another interpretation that models $P_2 \cup P$, and hence K models P_2.

Case 1. $I(a)=2$. Then $a \in P \subset P_2 \cup P$. But $K(a) \neq 2$ will imply $K(P_2 \cup P) \neq 2$ contradicting the fact that K models $P_2 \cup P$. So if $I(a)=2$ then $K(a)=2$.

Case 2. $I(a)=0$. Then $\neg a \in P \subset P_2 \cup P$. But $K(a) \neq 0$ will imply $K(\neg a) \neq 2$ and $K(P_2 \cup P) \neq 2$ contradicting the fact that K models $P_2 \cup P$. So if $I(a)=0$ then $K(a)=0$.

Case 3a. $I(a)=1$ and $K(a)=0$. Then $(\neg a \rightarrow a) \in P \subset P_2 \cup P$. But K will evaluate $K(\neg a \rightarrow a)=0$ and $K(P_2 \cup P)=0$ arising contradiction again.

Case 3b. $I(a)=1$ and $K(a)=1$. Then, if exists, take another atom b such that $I(b)=1$. Now $\{a \rightarrow b, b \rightarrow a\} \subset P \subset P_2 \cup P$ and, since K models $P_2 \cup P$, $K(a \leftrightarrow b)=2$, hence $K(a)=K(b)=1$. So in this case $I(a)=1$ implies $K(b)=1$ for all atoms b, leading to $I=K$. But, from hypothesis, I did not model P_2 and K does. Contradiction.

Case 3c. Previous two cases state $I(a)=1$ implies $K(a)=2$ and, together with cases 1 and 2, are sufficient to imply $K=I'$. So, as claimed, I' is the only model for $P_2 \cup P$. □

Proof of Theorem 5.5

Let P_1 and P_2 be two logic programs. If P_1 and P_2 are equivalent in G_3 then, for any P, $P_1 \cup P$ and $P_2 \cup P$ are equivalent in G_3. We will prove assuming that M is a min-answer set of $P_1 \cup P$ that it is also a min-answer set of $P_2 \cup P$. Since $P_1 \equiv_{G_3} P_2$ it is immediate that M is an answer set of $P_2 \cup P$ by Theorem 3.1.

Now, since M is a minimal model of $P_1 \cup P$, $P_1 \cup P \cup \neg\widetilde{M} \Vdash_C M$ but, in particular, $P_1 \cup P \equiv_C P_2 \cup P$ and therefore $P_2 \cup P \cup \neg\widetilde{M} \Vdash_C M$. By Lemma 5.1, M is a minimal model of $P_2 \cup P$. The same argument proves that every min-answer set of $P_2 \cup P$ is a min-answer set of $P_1 \cup P$. So the two programs are strongly equivalent with respect to the sematic of the min-answer sets.

For the converse, suppose that P_1 and P_2 are not equivalent in G_3 then, by Remark 5.6, there is an interpretation that models P_1, but not P_2 which, by Proposition 5.7, implies they are not strongly equivalent with respect to the semantic of the min-answer sets. □

References

Baral, C. 2003. *Knowledge Representation, reasoning and declarative problem solving with Answer Sets.* Cambridge University Press.

Bell, C., Nerode, A., Ng, R. and Subrahmanian, V. S. 1993. Implementing stable semantics by linear programming. In: L. M. Pereira and A. Nerode, Eds. *Proceedings of the Logic Programming and Non-Monotonic Reasoning 1993*, pp. 23–42. MIT Press.

Brouwer, L. E. J. 1981. *Brouwer's Cambridge Lectures on Intuitionism.* Cambridge University Press.

Dix, J., Osorio, M. and Zepeda, C. 2001. A general theory of confluent rewriting systems for logic programming and its applications. *Annals of Pure and Applied Logic 108*, 1–3, 153–188.

Erdem, E. and Lifschitz, V. 2001. Fages' theorem for programs with nested expressions. *Proceedings of the 17th International Conference on Logic Programming*, pp. 242–254. Springer, Paphos, Cyprus.

Eshgi, K. and Kowalski, R. 1989. Abduction compared with negation by failure. In: G. Levi and M. Martelli, Eds. *Logic Programming, Proceedings of the Sixth International Conference*, pp. 234–255. MIT Press.

Gelfond, M. and Lifschitz, V. 1990. Logic programs with classical negation. In: D. H. D. Warren and P. Szeredi, Eds. *Logic Programming, Proceedings of the Seventh International Conference*, pp. 579–597. MIT Press.

Gelfond, M., Przymusinska, H. and Przymusinski, T. 1989. On the relationship between circumscription and negation as failure. *Artificial Intelligence 38*, 75–94.

Girard, J.-Y. 1987. Linear logic. *Theoretical Computer Science 50*, 1–102.

Janhuenen, T. 2001. On the effect of default negation on the expressiveness of disjunctive rules. In: T. Eiter, W. Faber and M. Truszczynski, Eds. *Logic Programming and Nonmonotonic Reasoning, 6th International Conference, Lecture Notes in Computer Science 2173*, pp. 93–106. Springer-Verlag.

Kowalski, R. 2001. Is logic really dead or just sleeping. *Proceedings of the 17th International Conference on Logic Programming*, pp. 2–3. Springer, Paphos, Cyprus.

Liberatore, P. 1999. Algorithms and experiments on finding minimal models. Technical Report 09-99, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza".

Lifschitz, V., Pearce, D. and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic 2*, 526–541.

Lifschitz, V. and Schwarz, G. 1993. Extended logic programs as autoepistemic theories. In: L. M. Pereira and A. Nerode, Eds. *2nd International Workshop on Logic Programming & Non-Monotonic Reasoning*, pp. 101–114. MIT Press.

Lifschitz, V., Tang, L. R. and Turner, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25*, 369–389.

Lloyd, J. W. 1987. *Foundations of Logic Programming*, Second ed. Springer-Verlag.

Lobo, J. and Subrahmanian, V. S. 1992. Relating minimal models and pre-requisire-free normal defaults. *Information Processing Letters 44*, 129–133.

Marek, V. W. and Remmel, J. B. 2001. On the foundations of answer set programming. *Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, pp. 124–131. AAAI Press.

Marek, V. W. and Truszczyński, M. 1993. Reflexive autoepistemic logic and logic programming. In: L. M. and A. Nerode, Eds. *2nd International Workshop on Logic Programming & Non-Monotonic Reasoning*, pp. 115–131. MIT Press.

Minker, J. and Perlis, D. 1985. Computing protected circumscription. *The Journal of Logic Programming 2*, 235–249.

Navarro, J. A. 2002. Answer set programming through G_3 logic. In: M. Nissim, Ed. *Seventh ESSLLI Student Session, European Summer School in Logic, Language and Information*, Trento, Italy.

Niemelä, I. and Simons, P. 1996. Efficient implementation of the well-founded and stable model semantics. In: M. Maher, Ed. *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pp. 289–303. MIT Press.

Osorio, M. and Jayaraman, B. 1999. Aggregation and negation-as-failure. *New Generation Computing 17*, 3, 255–284.

Osorio, M., Jayaraman, B. and Plaisted, D. 1999. Theory of partial-order programming. *Science of Computer Programming 34*, 3, 207–238.

Osorio, M., Navarro, J. A. and Arrazola, J. 2001. Equivalence in answer set programming. In: A. Pettorossi, Ed. *Logic Based Program Synthesis and Transformation. 11th International Workshop, LOPSTR 2001: Lecture Notes in Computer Science 2372*, pp. 57–75. Springer-Verlag.

Osorio, M., Navarro, J. A. and Arrazola, J. 2002a. Answer set programming and S4. Unpublished.

Osorio, M., Navarro, J. A. and Arrazola, J. 2002b. Debugging in A-Prolog: A logical approach (abstract). In: P. Stuckey, Ed. *Logic Programming. 18th International Conference, ICLP 2002: Lecture Notes in Computer Science 2401*, pp. 482–483. Springer-Verlag.

Osorio, M., Navarro, J. A. and Arrazola, J. 2002c. A logical approach for A-Prolog. In: R. de Queiroz, L. C. Pereira, and E. H. Haeusler, Eds. *9th Workshop on Logic, Language, Information and Computation (WoLLIC): Electronic Notes in Theoretical Computer Scienc 67*, pp. 265–275. Elsevier.

Osorio, M., Palacios, J. J. and Arrazola, J. 2002. Towards a framework for answer set programming as provability in linear logic. In: J. B. Diaz, Ed. *Proceedings of the first IDEIA workshop (in conjuntion with IBERAMIA 2002)*, Sevilla, Spain.

Pearce, D. 1999a. From here to there: Stable negation in logic programming. In: D. M. Gabbay and H. Wansing, Eds. *What Is Negation?*, pp. 161–181. Kluwer.

Pearce, D. 1999b. Stable inference as intuitionistic validity. *Logic Programming 38*, 79–91.

Pearce, D., Sarsakov, V., Schaub, T., Tompits, H. and Woltran, S. 2002. A polynomial translation of logic programs with nested expressions into disjunctive logic programs: Preliminary report. In: P. J. Stuckey, Ed. *Logic Programming. 18th International Conference, ICLP 2002: Lecture Notes in Computer Science 2401*, pp. 405–420. Springer-Verlag.

Sakama, C. and Inoue, K. 1998. Negation as failure in the head. *Journal of Logic Programming 35(1)*, 39–78.

Troelstra, A. S. and van Dalen, D. 1988. *Constructivism in Mathematics: An introduction.* Vol. II. North-Holland.

van Dalen, D. 1980. *Logic and Structure*, Second ed. Springer-Verlag.

Zakharyaschev, M., Wolter, F. and Chagrov, A. 2001. Advanced modal logic. In: D. M. Gabbay and F. Guenthner, Eds. *Handbook of Philosophical Logic*, Second ed. Vol. 3, pp. 83–266. Kluwer Academic.