Available online at www.sciencedirect.com

SCIENCE DIRECT®

Annals of Pure and Applied Logic 134 (2005) 63–82

ANNALS OF PURE AND APPLIED LOGIC

www.elsevier.com/locate/apal

Safe beliefs for propositional theories

Mauricio Osorio[*], Juan Antonio Navarro Pérez, José Arrazola

Universidad de las Américas, CENTIA, Santa Catarina Mártir, Cholula, 72820 Puebla, Mexico

Received 17 December 2002; received in revised form 10 May 2004; accepted 23 June 2004
Available online 18 November 2004

Abstract

We propose an extension of answer sets, that we call safe beliefs, that can be used to study several properties and notions of answer sets and logic programming from a more general point of view. Our definition, based on intuitionistic logic and following ideas from D. Pearce [Stable inference as intuitionistic validity, Logic Programming 38 (1999) 79–91], also provides a general approach to define several semantics based on different logics or inference systems.

We prove that, in particular, intuitionistic logic can be replaced with any other proper intermediate logic without modifying the resulting semantics. We also show that the answer set semantics satisfies an important property, the "extension by definition", that can be used to construct program translations. As a result we are able to provide a polynomial translation from propositional theories into the class of disjunctive programs.

1. Introduction

Safe beliefs is an extension of the answer set ("stable model") semantics [6] that allows the use of arbitrary propositional theories as logic programs. The official definition of answer sets for augmented programs, due to Lifschitz et al. [10], is based in finding minimal models of some reduced logic programs. Studies from several authors in the context answer sets have provided interesting relations of this semantic operator with monotonic propositional logics.

[*] Corresponding author.
E-mail address: josorio@mail.udlap.mx (M. Osorio).

0168-0072/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.apal.2004.06.012

Work that relates answer sets with classical logic can be found in [5,15]. Erdem and Lifschitz relate answer sets and supported models in [4]. Work that relates answer sets with epistemic and modal logic can be found in [9,11]. Our work follows an approach, originally proposed by Pearce [22], that makes use of intuitionistic logic. Further research from Pearce [8,21] served to develop the equilibrium logic that can also be considered as an extension of answer sets.

Safe beliefs are then defined as extensions of theories satisfying certain consistency and completeness conditions. The use of intuitionistic logic as the underlying inference system seems appropriate since, according to Brouwer, we can identify a formula A with "I know A". Non-monotonic inference can be achieved determining some formulas that we can "safely believe" in order to produce new knowledge. The actual definition of safe beliefs, given later in this paper, formalizes the idea.

As a major advantage of this logic based approach we can study common notions and properties of answer sets from a more general point of view. We do not need, for instance, any particular assumptions about the form or syntax that logic programs should have. The definition of answer sets is automatically extended to deal with arbitrary propositional theories. Moreover, other propositional logics, even modal or temporal logics, can be used as the underlying inference system to produce different semantic operators with additional or extended features.

One of the main theorems of this paper shows that, in particular, any propositional logic, strictly weaker than classical logic and stronger than intuitionistic logic, can be used in the definition of safe beliefs without modifying the resulting semantic operator. Several applications and theoretical consequences of this result are also discussed. This result can be used to show that, in fact, our extension of answer sets is equivalent to the one provided by Pearce's equilibrium logic.

We also devote one section to the study of translations for logic programs under this generalized approach. We explore the notion of conservative transformations and, extending ideas from [8], we propose a definition of a *strong* conservative transformation. We show for instance that adding new atoms "by definition" to logic programs is an operation that preserves the semantics in a strong way. We also observe that this is an important property of answer sets that should be considered when studying properties of other semantics.

Another contribution of this paper is to present a translation, with this sort of strong properties, that can be used to reduce any propositional theory into the class of augmented programs. Other translations, like the ones presented in [23], can be used to provide a polynomial reduction of arbitrary propositional theories into disjunctive logic programs.

The material included in this paper is an extension and revision of earlier work [13,14, 18] already presented in workshops and conferences.

Our paper is structured as follows: in Section 2 we briefly present the language of propositional logic and several classes of logic programs commonly found in literature. We also review some basic notions and definitions of intermediate logics and present some simple results. In Section 3 we introduce the semantics of answer sets and safe beliefs, we also discuss the relations between the two approaches. In Section 4 we present the first of our main theorems, the invariance of safe beliefs with respect to the underlying logic.

In Section 5 we explore the idea of strong conservative transformations and provide our translation of arbitrary theories into augmented programs.

2. Background

We review in this section some basic concepts and definitions used throughout this paper. First we introduce the language of propositional logic, and use it to define some classes of programs commonly found in literature. Finally we make some comments on intermediate logics that will be used in later sections to provide a logical framework for logic programming and non-monotonic reasoning.

2.1. Propositional logic

We use the complete set of propositional formulas in order to describe rules and information within logic programs. Formally we consider a language built from an alphabet containing: a denumerable set $\mathcal{L}$ of elements called *atoms* or *atomic formulas*; the 2-place connectives $\wedge$, $\vee$, $\to$; the 0-place connective $\bot$; and auxiliary symbols (,). Formulas are constructed as usual in logic. The formula $\top$ is introduced as an abbreviation of $\bot \to \bot$, $\neg A$ as an abbreviation of $A \to \bot$, and $A \leftrightarrow B$ as an abbreviation of $(A \to B) \wedge (B \to A)$. The notation $A \leftarrow B$ is just another way of writing the formula $B \to A$.

A *theory* is a set of formulas, we restrict our attention to finite theories. For a given theory T its *signature*, denoted $\mathcal{L}_T$, is the set of atoms that occur in the theory T. Observe that, since we consider finite theories, their signatures are also finite. A *literal* is either an atom a (a positive literal) or a negated atom $\neg a$ (a negative literal). Given a theory T we also define the negated theory $\neg T = \{\neg A \mid A \in T\}$.

2.2. Logic programs

In our context a *logic program* is just a propositional theory, and a *class of logic programs* is a set of logic programs. We can think, in fact, of the words *theory* and *program* as synonyms; we use the former when stating general results in mathematical logic, and the latter when dealing with logic programs having a particular syntax and/or semantics.

We use the symbol Prp to denote the class containing all propositional logic programs. A *positive theory* (or program) is one that does not contain occurrences of the connective $\bot$ (except for those in the formula abbreviated by $\top$), we use Pos to denote the set of all positive theories.

The syntax of formulas within logic programs is usually defined in terms of special formulas know as clauses. A *clause* is, in general, a formula $H \leftarrow B$ with implication as the principal connective. The formulas H and B are known as the *head* and *body* of the clause respectively. The special case of a clause with the form $\bot \leftarrow B$ is known as a *constraint* and the head is said to be empty. Each formula H, whose principal connective is not implication, is know as a *fact* and is associated with the clause $H \leftarrow \top$ that has an empty body.

We introduce now some of the classes of programs commonly found in literature. An *augmented clause* is a clause where both H and B can be arbitrary logic formulas built from the connectives $\wedge$, $\vee$ and $\neg$ arbitrarily nested. Note, however, that embedded implications are not allowed in augmented clauses. An *augmented program* is a logic program that contains only augmented clauses, and the class of all augmented programs is denoted Aug.

A *disjunctive clause* is built from a (non-empty) disjunction of atoms in the head and a conjunction of literals in the body. A disjunctive clause has then the form

$$h_1 \vee \cdots \vee h_n \leftarrow b_1 \wedge \cdots \wedge b_m \wedge b_{m+1} \wedge \cdots \wedge b_{m+l}$$

where each h_i and b_j is an atom, $n \geq 1$, $m \geq 0$ and $l \geq 0$. A *disjunctive program* is then defined as a program containing only disjunctive formulas, and the class of all disjunctive programs is denoted Dis. Observe that, in particular, constraints are not allowed inside disjunctive programs.

Example 1. The following are examples of the clauses just defined

$a \leftarrow (b \rightarrow c).$	propositional
$\neg(p \wedge \neg q) \leftarrow a \vee (\neg b \wedge c).$	augmented
$a \vee b \leftarrow c \wedge d \wedge \neg e.$	disjunctive

Also observe that these classes of programs satisfy Dis $\subset$ Aug $\subset$ Prp.

2.3. *Basic notions on intermediate logics*

This paper follows a line of research aimed to provide foundations of logic programming in terms of mathematical logic. We found that intermediate logics are particularly useful for this approach. Intuitionistic and multivalued logics are briefly described in the following paragraphs. A more complete background on these logics can be found in [26]. Some notation, definitions and simple results are given at the end of this section.

2.3.1. *Intermediate logics*

Some logics can be defined in terms of truth values and evaluation functions. Gödel defined the multivalued logics G_i, with i truth values, where a *model* of a formula is a truth assignment to the atoms that, when propagated over its connectives, evaluates to some designated *true* value. A formula is a *tautology* if it is true for every possible truth assignment. Observe that G_2 corresponds exactly to the definition of classical logic C.

Another important logic, which has been a great area of interest in recent years, is intuitionistic logic. This logic is based on the concept of *proof*, rather than *truth* as in classical logic. Intuitionistic logic, denoted I, can be defined in terms of Hilbert type proof systems of axioms and inference rules where the provable formulas are known as *theorems*. Equivalent definitions can be given in terms of natural deduction systems and Kripke models, see [24,25]. Surprisingly no definition using a truth table scheme is possible [12].

Gödel observed that there are infinitely many logics whose set of provable formulas lies between intuitionistic and classical logic [26]. We have, in particular, the following proposition, where $\subset$ denotes proper contention of the set of provable formulas (theorems or tautologies) on each logic.

Proposition 2 (*[26]*). $\mathrm{I} \subset \cdots \subset \mathrm{G}_{i+1} \subset \mathrm{G}_i \subset \cdots \subset \mathrm{G}_3 \subset \mathrm{G}_2 = \mathrm{C}$.

Any logic[1] whose set of provable formulas lies between intuitionistic and classical logic (inclusive), are known as *intermediate logics*. A *proper intermediate logic* is an intermediate logic that is not the classical one. Proposition 2 shows that the multivalued logics G_i are all intermediate logics.

The logic G_3, in particular, corresponds to the strongest proper intermediate logic [26]. Alternate definitions for the logic G_3 are also available. The *Here and There*, HT, logic defined using Kripke models with two worlds is equivalent to the logic G_3. Following an axiomatic approach the logic Sm, which is obtained adding the new axiom scheme

$$(\neg q \to p) \to (((p \to q) \to p) \to p)$$

to the set of intuitionistic axioms, is equivalent to the logic G_3.

Proposition 2 also seems to imply that intuitionistic logic is less powerful than classical logic since, in fact, it proves “less” formulas. Gödel showed, however, that the provable formulas of classical logic are actually embedded inside intuitionistic logic. A slightly more general version of this result is presented later in Proposition 3.

Intuitionistic logic, as well as the proper intermediate logics, are able to distinguish between a and $\neg\neg a$. This property, as we will see later, makes these logics suitable to characterize notions of logic programming.

2.3.2. General definitions

We use the standard notation $\vdash_{\mathrm{X}} F$ to denote that F is a provable formula (a tautology or a theorem) of logic X. If T is a theory we understand the symbol $T \vdash_{\mathrm{X}} F$ to mean that $\vdash_{\mathrm{X}} (F_1 \wedge \cdots \wedge F_n) \to F$ for some formula 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*. Also, for a given theory T, the set $Cn_{\mathrm{X}}(T) = \{F \mid T \vdash_{\mathrm{X}} F\}$ is known as the set of X-consequences of T.

Using this notation the result about classical logic embedded inside intuitionistic logic can be presented as follows.

Proposition 3 (*[19]*). *Let T be a theory and let F be a formula. $T \vdash_{\mathrm{C}} F$ if and only if $T \vdash_{\mathrm{I}} \neg\neg F$.*

Similarly, if U is a theory, we use the symbol $T \vdash_{\mathrm{X}} U$ to denote $T \vdash_{\mathrm{X}} F$ for every $F \in U$. Then we can say that two theories T_1 and T_2 are equivalent, with respect to logic X and denoted $T_1 \equiv_{\mathrm{X}} T_2$, if it is the case that both $T_1 \vdash_{\mathrm{X}} T_2$ and $T_2 \vdash_{\mathrm{X}} T_1$. Observe that $T_1 \equiv_{\mathrm{X}} T_2$ if and only if $Cn_{\mathrm{X}}(T_1) = Cn_{\mathrm{X}}(T_2)$.

A theory T is said to be X-consistent if it is not the case that $T \vdash_{\mathrm{X}} \bot$. It is easy to show, see for instance [25], that the definition does not depend on the particular logic chosen, so we can drop the prefix “X-” from the notation.

[1] A *logic* in the propositional case can be defined as a set of propositional formulas closed under logical consequence (i.e. *modus ponens* and *substitution*).

Lemma 4 (*[25]*). *Let* X *and* Y *be two intermediate logics, and let* T *be a theory. Then* T *is* X*-consistent if and only if it is* Y*-consistent.*

For a theory T and a formula F we say that F is X-*determined* by T if it is the case that either $T \vdash_{\mathrm{X}} F$ or $T \vdash_{\mathrm{X}} \neg F$. A theory is said to be X-*complete* if every formula F, with $\mathcal{L}_F \subseteq \mathcal{L}_T$ is X-determined by the theory. Similarly, a theory is *literal* X-*complete* if every atom in $\mathcal{L}_T$ is X-determined. It is clear that completeness implies literal completeness. The converse is also true.

Lemma 5. *Let* X *be an intermediate logic and let* T *be a theory. If* T *is literal* X*-complete then* T *is also* X*-complete.*

Proof. Observe, since T is literal X-complete we can define a unique classical two valued interpretation I such that $I(a) = 1$ if $T \vdash_{\mathrm{X}} a$ and $I(a) = 0$ otherwise. Then for any formula F it is easy to verify, using structural induction over the connectives of F, that $T \vdash_{\mathrm{X}} F$ if I is a model of F and $T \vdash_{\mathrm{X}} \neg F$ otherwise. □

We end this section with a simple lemma that formally presents an important property of intermediate logics that we will freely use in the following sections. A proof in the context of intuitionistic logic is given in [25], but this same proof holds under any other intermediate logic.

Lemma 6. *Let* P *be a theory and let* A, B *be a pair of formulas. If* P' *is a theory obtained from* P *by replacing some occurrences of* A *with* B, *and* T *is a theory such that* $T \vdash_{\mathrm{X}} A \leftrightarrow B$, *for an intermediate logic* X, *then* $T \cup P \equiv_{\mathrm{X}} T \cup P'$.

3. Semantics for logic programs

We use semantics to assign a meaning to a given logic program. Given a class of logic programs C, a *semantic operator* Sem is a function that assigns to each program $P \in C$ some sets of atoms $M \subseteq \mathcal{L}_P$. These sets of atoms are usually some "preferred" models of the logic program P.

A popular semantic operator for logic programs is the answer set semantics. Some of the main features of answer sets are its non-monotonicity and the integration of negation as failure.[2] These sorts of features are extremely useful to deal with concepts such as default knowledge and modeling inertial rules. The answer set semantics constitutes the formal basement of the A-Prolog programming paradigm and proved to be particularly useful for several real life and research applications.

The actual definition of answer sets is, however, not required for the purposes of this paper, the reader is referred to [10] for more details. It is just important to mention that, from the original formal definition, the relation of the semantics with a particular logic is not entirely clear and seems to be hidden under an ad hoc setting of reductions and minimal models.

[2] The negation symbol we use ($\neg$) will play the role of the negation as failure in logic programs. The authors in [10] use, however, the symbol $\sim$ instead.

3.1. A logical approach

Consider the case of a logic agent. The task of such an agent is, when provided with some initial knowledge describing its world or application domain, to do inference and produce new knowledge. We would like our agent, for instance, to be able to answer queries and solve problems in its application domain.

We can use a propositional theory T to represent the initial knowledge of our agent. Under the premise that T is consistent, it makes sense to say that the agent *knows* F if the formula F is an intuitionistic consequence of the theory T. The use of intuitionistic logic, a logic of knowledge, as the underlying inference system seems a natural choice for this approach.

But we also want our agent to do non-monotonic inference. Informally speaking we will allow our agent to *guess* or *suppose* things in order to make more inference. We must be cautious, however, since we don't want our agent to precipitate conclusions or make unnecessary assumptions. The agent should only *suppose* facts if they are helpful to provide new knowledge. The following definition formalizes this idea.

Definition 7. Let T be a theory and let E be a theory with $\mathcal{L}_E \subseteq \mathcal{L}_T$. We say that the theory E is an I-*explanation* of T if the theory $T \cup \neg\neg E$ is both consistent and I-complete.

A very natural reading of this definition, in the context of our logic agent, would be that: *"A theory E is an explanation of the agent's world if* (i) *it is consistent with its initial knowledge and* (ii) *supposing that E is true is enough to answer any possible question in its domain"*. This *supposing* corresponds to the double negation in the intuitionistic proof system used as the underlying inference system.

The agent is allowed now to *believe* things in order to obtain a complete explanation of its world. If E is an I-explanation of T, it makes sense to say that the agent *believes* F (assuming E) if the formula F is an intuitionistic consequence of E. There can also be, of course, several explanations for a given initial knowledge. Brave and cautious beliefs can be defined in the natural way. The agent *bravely believes* a formula if there is an explanation that makes the formula true, and *cautiously believes* a formula if that formula is true under all possible explanations of its initial knowledge.

The explanations for a given theory naturally define semantics for logic programs, we call this the (intuitionistic) *safe belief semantics*.

Definition 8. Let T be a theory. If E is an I-explanation of T then the set of atoms $Cn_{\mathrm{I}}(T \cup \neg\neg E) \cap \mathcal{L}_T$ is an I-*safe belief* of T.

Observe that this definition does not impose any particular restriction in the form or syntax that the logic programs should have. In our particular case this definition allows the use of embedded implications inside formulas, a feature that is not very common in current logic programming paradigms. The use of the full set of propositional formulas can be useful to describe problems in a more convenient and natural way.

Another important property of safe beliefs is that its definition naturally follows from provability relations in a well known mathematical logic, namely intuitionistic logic, providing solid foundations for our approach. The relation with logic is more clear and direct than, for instance, in the original definition of answer sets.

3.2. Logical foundations for A-Prolog

Despite the apparent lack of a logical intuition in the definition of answer sets, it is possible to show that they actually coincide with safe beliefs (in the class of augmented programs). This is not only a significant property of safe beliefs, but also serves to provide solid logical foundations for the A-Prolog programming paradigm and allows natural generalizations.

The use of a logic based approach to study answer sets has been proved to be quite useful in several circumstances. The study of notions such as strong equivalence between logic programs is closely related with equivalence in intermediate logics [8,13]. Other interesting applications, such as the definition of a debugging scheme for logic programming, is also possible by taking advantage of the 3-valued nature of the logic G_3 [1,17].

Definition 8 follows from a line of research, initiated by Pearce [22], that offered a characterization of answer sets in terms of intuitionistic logic. Pearce proposed an approach based on completions of theories, very similar to our explanations but based on the logic HT,[3] and showed that they are equivalent to the *equilibrium logic* also developed by himself [21,22]. The *equilibrium models* of this logic, which are defined for arbitrary propositional theories, are some distinguished HT models satisfying certain minimality conditions.[4] We can rephrase this result as follows:

Proposition 9 (*[22]*). *Let T be any theory. A theory E is an* HT-*explanation of the theory T if and only if the set of atoms $Cn_{\mathrm{HT}}(T \cup \neg\neg E) \cap \mathcal{L}_T$ is an equilibrium model of E.*

Pearce also investigated intuitionistic explanations, but restricted to explanations composed exclusively by negated atoms, and he used them to characterize answer sets of disjunctive logic programs [22]. His characterization, however, was not intended to characterize answer sets for other classes of programs. In fact the characterization does not capture correctly the answer sets of programs containing negation in the head [19]. In a more recent paper [8], together with Lifschitz and Valverde, Pearce was able to show that equilibrium models actually characterize answer sets for augmented programs, the most general class of programs where a definition for answer sets is available.

Proposition 10 (*[8]*). *For any augmented program P and any set M of atoms, the set M is an equilibrium model of P iff M is an answer set of P.*

Recall that the definition of safe beliefs is based on the inference system provided by intuitionistic logic, but analogous definitions can be naturally defined for any other propositional logic. Proposition 9 deals, for instance, with the case of HT-explanations. An interesting question left open is whether different logics produce different semantics for logic programs, or if all of them collapse to the same non-monotonic inference system.

If we use classical logic the answer is immediate and quite simple, classical safe beliefs correspond just to regular monotonic classical models. But the situation is not that clear

[3] They are introduced in a somewhat different manner, but a close inspection shows that the difference is just a matter of notation.

[4] The formal definition of *equilibrium models* is not required for the purposes of this paper. We mention them just to state the relations between different approaches to the study of answer sets.

for the rest of the proper intermediate logics. One of the main contributions of this paper is to finally answer this question. In Section 4 we prove that all proper intermediate logics capture exactly the same models as safe beliefs.

The characterization of answer sets in terms of I-explanations, which was originally presented in [19], is now rediscovered as a simple consequence of this result. Therefore, at the end of Section 4, we can prove that the semantics of answer sets are equivalent to I-safe beliefs for augmented programs, providing logical foundations for the A-Prolog paradigm.

4. Safe beliefs for intermediate logics

This section is concentrated in the proof of one of the main results of this paper, namely the invariance of safe beliefs with respect to different proper intermediate logics. In order to do so we present first some preliminary lemmas that are used in the main proof. We define then some reductions of theories that, as a theoretical application, are quite useful to complete our proof. At the end of this section we present the formal proof of our statement.

4.1. Preliminaries

The following is a particular case of a proposition in [16]. The original proposition presents a class of formulas that are irrelevant in the proof of a positive formula from positive premises. We use this result to remove premises of the form $\neg\neg M$, for a set M of atoms, when we are dealing with proofs of positive formulas. Recall the definition of *positive theories* from Section 2.2.

Lemma 11 ([16]). *Let T be a positive theory, let M be a set of atoms and let F be a positive formula. If $T \cup \neg\neg M \vdash_{\mathrm{I}} F$ then $T \vdash_{\mathrm{I}} F$.*

The next lemma shows a particular case where provability relations of classical and intuitionistic logics coincide. Namely, if a theory has the property that it can prove true – using classical logic – all the atoms in its own signature, then the use of intuitionistic logic will preserve this property of the theory.

Lemma 12. *Let T be a positive theory. If $T \vdash_{\mathrm{C}} \mathcal{L}_T$ then $T \vdash_{\mathrm{I}} \mathcal{L}_T$.*

Proof. Let a be an atom in $\mathcal{L}_T$. We can replace every atom in the proof of $T \vdash_{\mathrm{C}} a$ to obtain a valid proof of $T_a \vdash_{\mathrm{C}} a$ where the theory T_a is obtained by replacing all atoms occurring in T with the atom a. Obtain now the program T'_a applying the following replacements on T_a until no more replaces can be done:

- $\top \wedge \top$ with $\top$.
- $a \vee a$ with a.
- $\top \to a$ with a.
- $a \wedge a, a \wedge \top, \top \wedge a$ with a.
- $a \vee \top, \top \vee a, \top \vee \top$ with $\top$.
- $a \to a, a \to \top, \top \to \top$ with $\top$.

It is clear that $T_a \equiv_{\mathrm{X}} T'_a$ for any intermediate logic X. Also observe that each formula in T_a must be reduced either to a or $\top$, i.e. $T'_a \subseteq \{\top, a\}$. Since we must have $T'_a \vdash_{\mathrm{C}} a$ then it must be the case that $a \in T'_a$. It is now an immediate consequence that $T'_a \vdash_{\mathrm{I}} a$ and, restoring the equivalent theory T_a, $T_a \vdash_{\mathrm{I}} a$.

Now let $D = \{x \leftrightarrow y \mid x, y \in \mathcal{L}_T\}$, it follows that $T_a \cup D \vdash_{\mathrm{I}} a$ and, since $T_a \cup D \equiv_{\mathrm{I}} T$, we may conclude that $T \vdash_{\mathrm{I}} a$. □

The following lemma shows that we can eliminate premises whose language has no relation with the formula that we want to prove.

Lemma 13. *Let T and U be two theories, and let F be a formula such that $\mathcal{L}_{T\cup\{F\}} \cap \mathcal{L}_U = \emptyset$. If the theory U is consistent and $T \cup U \vdash_{\mathrm{X}} F$, for some intermediate logic* X, *then $T \vdash_{\mathrm{X}} F$.*

Proof. Since the theory U is consistent it has, at least, one classical model M. In the proof of $T \cup U \vdash_{\mathrm{X}} F$ we can replace each atom $a \in \mathcal{L}_U$ with $\top$ if $a \in M$ and with $\bot$ if $a \notin M$. Since T and F do not contain any of the atoms $a \in \mathcal{L}_U$, these formulas are not modified. While all the formulas in U are replaced with theorems or tautologies from the underlying logic. This leaves a proof for $T \vdash_{\mathrm{X}} F$. □

4.2. Reduction of theories

In this section we present some reductions of theories and several of their properties in terms of intermediate logics. The notion of reductions and/or transformations has several applications in logic programming; see for instance [3,16,20]. Our set of reductions can be used, in particular, as a theoretical tool in the proof of Theorem 25.

The first of these reductions evaluates the effect of the connectives $\bot$ and $\top$[5] inside a logic program. After applying this reduction all occurrences of $\bot$ will be removed, except for those that appear in the form $F \to \bot$ as a negation or if the entire theory is reduced to the single connective $\bot$. Besides, no occurrence of the connective $\top$ will remain in the program.

Definition 14. Given a formula F we define its reduction with respect to $\bot$, denoted $\mathrm{Reduce}_\bot(F)$, as the formula obtained applying the following replacements on F until no more replaces can be done. If A is any formula then replace

- $A \wedge \top$ or $\top \wedge A$ with A.
- $A \vee \top$ or $\top \vee A$ with $\top$.
- $A \to \top$ or $\bot \to A$ with $\top$.
- $A \wedge \bot$ or $\bot \wedge A$ with $\bot$.
- $A \vee \bot$ or $\bot \vee A$ with A.
- $\top \to A$ with A.

Then, for a given theory T, we define $\mathrm{Reduce}_\bot(T) = \{\bot\}$ if there is any formula $F \in T$ such that $\mathrm{Reduce}_\bot(F) = \bot$ and, otherwise,

$$\mathrm{Reduce}_\bot(T) = \{\mathrm{Reduce}_\bot(F) \mid F \in T \text{ and } \mathrm{Reduce}_\bot(F) \neq \top\}.$$

It follows immediately from the definition that the reduction with respect to $\bot$ of any theory is equivalent, under any intermediate logic, to the original theory.

Proposition 15. *For any theory T and any intermediate logic* X, *we have that $T \equiv_{\mathrm{X}} \mathrm{Reduce}_\bot(T)$.*

[5] Formally $\top$ is not a connective but an abbreviation of the formula $\bot \to \bot$. It is useful, in order to keep the presentation simple, to assume that $\top$ is in fact a connective.

Proof. It suffices to observe that each one of the replacements preserve equivalence with respect to intuitionistic logic. They are valid, in particular, in any intermediate logic. □

The next reduction works, given a set M of atoms, assuming that the atoms contained in M are false (i.e. assuming $\neg M$) and reducing a theory using this information. One of the important properties of this reduction is that, after applying it, the reduced theory will not contain any of the atoms in M. The reduction also preserves equivalence under intermediate logics appending the set of premises $\neg M$.

Definition 16. Let T be a theory and let M be a set of atoms. Let T' be the theory obtained replacing each occurrence of atoms $a \in M$ in T with the connective $\bot$. Then we define the reduction $\text{Reduce1}(T, \neg M) = \text{Reduce}_\bot(T')$.

Proposition 17. *For any theory T, any intermediate logic* X *and any set M of atoms, we have that $T \cup \neg M \equiv_{\mathrm{X}} \text{Reduce1}(T, \neg M) \cup \neg M$.*

Proof. It follows from Proposition 15 and the fact that, for any atom a, we have that $\neg a \vdash_{\mathrm{X}} a \leftrightarrow \bot$. □

Our last reduction is similar to the previous one. Given a set M of atoms it evaluates the effect on a theory of adding the premises $\neg\neg M$. Since we are interested in preserving equivalence with respect to intermediate logics, the reduction can only take effect inside negated subformulas. Only atoms occurring in formulas $F \to \bot$ are replaced by the reduction.

Definition 18. Let T be a theory and let M be a set of atoms. Let T' be the theory obtained by replacing all occurrences of atoms $a \in M$ in T with the connective $\bot$ if the occurrence of the atom appears within a formula $F \to \bot$. Then we define the reduction $\text{Reduce2}(T, \neg\neg M) = \text{Reduce}_\bot(T')$.

Proposition 19. *For any theory T, any intermediate logic* X *and any set M of atoms, we have that $T \cup \neg\neg M \equiv_{\mathrm{X}} \text{Reduce2}(T, \neg\neg M) \cup \neg\neg M$.*

Proof. It also follows from Proposition 15 and the fact that for any formula F, and if we replace the occurrences of an atom a for $\top$ to obtain F', then we have that $\neg\neg a \vdash_{\mathrm{X}} \neg F \leftrightarrow \neg F'$. □

Example 20. This example illustrates the use of these reductions. Consider the following theory T:

$$\begin{array}{l} (a \wedge b) \vee \neg(c \wedge d). \\ b \wedge d \to b. \\ (c \to (a \to b)) \vee e. \end{array}$$

In order to compute $\text{Reduce1}(T, \{\neg b\})$ we have to replace all occurrences of the atom b with the connective $\bot$ and successively apply the $\text{Reduce}_\bot$ replacements.

$$\begin{array}{lclcl} (a \wedge \bot) \vee \neg(c \wedge d). & & \bot \vee \neg(c \wedge d). & & \neg(c \vee d). \\ \bot \wedge d \to \bot. & \Longrightarrow & \bot \to \bot. & \Longrightarrow & \top. \\ (c \to (a \to \bot)) \vee e. & & (c \to (a \to \bot)) \vee e. & & (c \to (a \to \bot)) \vee e. \end{array}$$

So that, after one more aplication of the reduction $\text{Reduce}_\perp$, the theory $T' = \text{Reduce1}(T, \{\neg b\})$ obtained is:

$$\begin{array}{l} \neg(c \wedge d). \\ (c \to (a \to \perp)) \vee e. \end{array}$$

If we compute now $T'' = \text{Reduce2}(T', \{\neg\neg a, \neg\neg c\})$ we start by replacing occurrences of a and c, that appear under the scope of a negation, with the connective $\top$ and we apply then $\text{Reduce}_\perp$:

$$\begin{array}{l} \neg(\top \vee d). \\ (c \to (\top \to \perp)) \vee e. \end{array} \Longrightarrow \begin{array}{l} \neg d. \\ (c \to \perp) \vee e. \end{array}$$

Observe now that, as a result of the application of Reduce2, a new negation has been produced in the last formula. We can therefore apply the reduction again to obtain $T''' = \text{Reduce2}(T'', \{\neg\neg a, \neg\neg c\}) = \{\neg d, e\}$.

As the previous example shows, in some cases the reduction Reduce2 can be applied several times over a theory to produce further simplifications. We use the symbol $\text{Reduce2}^*(P, \neg\neg M)$ to denote the theory obtained after successive applications of Reduce2 until no more reductions can be obtained. We can present now a reduction that integrates all previous ones.

Definition 21. Let T be a theory and let M_1, M_2 be two disjoint sets of atoms. We define $\text{Reduce}(T, \neg M_1, \neg\neg M_2) = \text{Reduce2}^*(\text{Reduce1}(T, \neg M_1), \neg\neg M_2)$.

Proposition 22. *For any theory T, any intermediate logic* X *and any pair of disjoint sets M_1, M_2 of atoms, we have that*

$$T \cup \neg M_1 \cup \neg\neg M_2 \equiv_{\mathrm{X}} \text{Reduce}(T, \neg M_1, \neg\neg M_2) \cup \neg M_1 \cup \neg\neg M_2 .$$

Proof. It follows from Propositions 17 and 19. $\square$

Also observe that, according to the notation of the previous proposition, if the sets satisfy $M_1 \cup M_2 = \mathcal{L}_T$ then the theory $\text{Reduce}(T, \neg M_1, \neg\neg M_2)$ must be a positive theory. This is because the reduction Reduce2 involved could be iteratively applied while negation is still occurring in the reduced theory.

4.3. Invariance of safe beliefs

One of the major contributions of this paper is to prove that intuitionistic logic, used to define the semantics of safe beliefs, can be replaced with any other intermediate logic without modifying the models that the semantics would capture as safe beliefs. The formal statement is given in Theorem 25 at the end of this section.

We need first the following simple lemma which is used to obtain, from every X-safe belief of a given theory, a "canonical" X-explanation of the theory. This canonical explanation has the property of being always X-complete and composed only by literals. These *canonical explanations* are useful to take advantage of the reductions we presented in the previous section.

Lemma 23. *Let T be a theory and let M be, for some intermediate logic* X*, an* X*-safe belief of T. The theory $E = M \cup \neg(\mathcal{L}_T \setminus M)$ is an* X*-explanation of T.*

Proof. If M is an X-safe belief of the theory T then, by definition, there must be a theory E' such that E' is an X-explanation of T and $M = Cn_{\mathrm{X}}(T \cup \neg\neg E') \cap \mathcal{L}_T$.

First we will show that $E \vdash_{\mathrm{X}} E'$. Suppose that there is a formula $F \in E'$ such that $E \nvdash_{\mathrm{X}} F$, since E is clearly X-complete then $E \vdash_{\mathrm{X}} \neg F$. Recall now that, by construction, $T \cup \neg\neg E' \vdash_{\mathrm{X}} E$ and, therefore, $T \cup \neg\neg E' \vdash_{\mathrm{X}} \neg F$. But this would imply that the theory $T \cup \neg\neg E'$ is inconsistent and contradiction arises.

It is clear, since E is consistent, that also $T \cup \neg\neg E$ is consistent. Finally, since $E \vdash_{\mathrm{X}} E'$, we have that $P \cup \neg\neg E \vdash_{\mathrm{X}} P \cup \neg\neg E'$. The two theories must have the same X-consequences and, in particular, $P \cup \neg\neg E$ is also X-complete. □

The following proposition solves one particular case of our main theorem, it shows that G_3-safe beliefs are also I-safe beliefs. As we can see in the proof of Theorem 25 this is the most important case since all other equivalences will follow from this one.

Proposition 24. *Let T be a theory, let M be a set of atoms with $M \subseteq \mathcal{L}_T$ and let $E = M \cup \neg(\mathcal{L}_T \setminus M)$. If the theory E is a G_3-explanation of T, then E is also an I-explanation of T.*

Proof. Since E is clearly consistent, we only need to show that $T \cup \neg\neg E$ is, in particular, literal I-complete. If we take an atom $a \notin M$ it is immediate, since we have $\neg a \in E$, that $T \cup \neg\neg E \vdash_{\mathrm{I}} \neg a$ (recall that $\vdash_{\mathrm{I}} \neg\neg\neg a \to \neg a$). We have only left to prove that, in fact, $T \cup \neg\neg E \vdash_{\mathrm{I}} M$.

By construction of E we already know that $T \cup \neg\neg E \vdash_{\mathrm{G}_3} M$. Let T' be the theory Reduce1$(T, \neg(\mathcal{L}_T \setminus M))$ so that, from Proposition 17, it follows that $T' \cup \neg\neg E \vdash_{\mathrm{G}_3} M$. Recall that the theory T' satisfies $\mathcal{L}_{T'} \subseteq M$ and, therefore, we can apply Lemma 13 to obtain $T' \cup \neg\neg M \vdash_{\mathrm{G}_3} M$.

We can, equivalently, assume that $T' \cup A \cup \neg\neg M \vdash_{\mathrm{I}} M$ where A is a set of instances of the axiom scheme

$$(\neg G \to F) \to (((F \to G) \to F) \to F) \tag{1}$$

that characterizes logic Sm. Without loss of generality, we may also assume that the theory A satisfies $\mathcal{L}_A \subseteq M$.

Let $T'' = \text{Reduce2}^*(T', \neg\neg M)$ and let $A' = \text{Reduce2}^*(A, \neg\neg M)$. By Proposition 19 we have that $T'' \cup A' \cup \neg\neg M \vdash_{\mathrm{I}} M$ where T'' and A' are positive theories. We can now, by Lemma 11, eliminate the set $\neg\neg M$ from the premises to obtain $T'' \cup A' \vdash_{\mathrm{I}} M$. In particular we also have that $T'' \cup A' \vdash_{\mathrm{C}} M$.

Now observe that, since Reduce2* completely evaluates the negation $\neg G$ in the axioms of the form (1), the set A' can only contain classical tautologies. We may then omit the set A' to finally obtain $T'' \vdash_{\mathrm{C}} M$. Observe now that we must have $\mathcal{L}_{T''} = M$ and, by Lemma 12, $T'' \vdash_{\mathrm{I}} M$. We can just restore $\neg\neg E$ as a set of premises and replace T'' with the original T to obtain $T \cup \neg\neg E \vdash_{\mathrm{I}} M$. □

We are now ready to prove one of the major results of our paper. As we anticipated, it shows that the definition of safe beliefs does not depend on the particular logic chosen as the underlying inference system.

Theorem 25. *Let T be a theory and let* X, Y *be two proper intermediate logics. A set M of atoms is an* X*-safe belief of T if and only if it is a* Y*-safe belief of T.*

Proof. If the set M is an X-safe belief of T then, by Lemma 23, the theory $E = M \cup \neg(\mathcal{L}_T \setminus M)$ is an X-explanation of T. Recall that G_3 is the strongest proper intermediate logic, then it follows that E is also a G_3-explanation of the theory T. Proposition 24 implies that E is an I-explanation of T and, since I is the weakest intermediate logic, E is a Y-explanation of T. Finally observe that, in fact, $M = Cn_Y(T \cup \neg\neg E) \cap \mathcal{L}_T$ and therefore M is a Y-safe belief of T. □

As an immediate consequence of the previous theorem we may drop the prefix "X-" from the notation of safe beliefs. We also have the following corollary that states the equivalence between the semantics of safe beliefs and answer sets.

Corollary 26. *Let P be an augmented program. A set of atoms M is a safe belief of P if and only if M is an answer set of P.*

Proof. By the previous theorem, we may assume that M is a G_3-safe belief of P. Now M is a G_3-safe belief if and only if, by Proposition 9, M is an equilibrium model of P if and only if, by Proposition 10, M is an answer set of P. □

5. Program translations

In this section we study several properties of translations for logic programs. It comprises a revision and extension of the material in [14]. We present first the notion of conservative transformations and, following ideas from [8], we also present the *strong* version of this notion. Using this concepts we are able then to show a polynomial time translation from propositional theories into the class of disjunctive logic programs.

5.1. Conservative transformations

Suppose that we have a logic program P and we want to compute its safe beliefs. The task could be simplified if we are able to construct another simpler logic program P', such that the safe beliefs of P and P' are somehow related. We could then compute the safe beliefs of P' and recover those of the original P. It is important, of course, that the "recovery" of the safe beliefs of P, knowing those of P', can be done through a simple and efficient method. A conservative transformation is a relation between logic programs that can be used for this kind of application.

Definition 27. Let Sem be a semantic operator defined for a class of logic programs C and let $P, P' \in C$. We say that P' is a *conservative transformation* of the program P, denoted $P \xrightarrow{\text{Sem}} P'$, if $\mathcal{L}_P \subseteq \mathcal{L}_{P'}$ and

$$\text{Sem}(P) = \left\{ M \cap \mathcal{L}_P \mid M \in \text{Sem}(P') \right\}.$$

A *conservative extension* is a similar notion, introduced in [2], with the additional condition $P \subseteq P'$ on the logic programs. Our definition is more general since it allows the program to be modified or "transformed", not only extended. The definition of a conservative transformation was originally presented in [14].

A conservative transformation of a logic program P can, possibly, introduce new atoms (those in $\mathcal{L}_{P'} \setminus \mathcal{L}_P$) in order to achieve simplifications. But, if we ignore these newly introduced atoms, we obtain exactly the answer sets of P. We refer to these new atoms in $\mathcal{L}_{P'} \setminus \mathcal{L}_P$ as the *reserved* atoms of the transformation. It is easy to verify that conservative transformations define a transitive relation.

One particular case of a conservative extension, in the context of safe beliefs, is useful to define new atoms as abbreviations of formulas in the current program domain. The extension of programs by adding definitions is an important, and certainly desirable, feature of the semantic of safe beliefs. We use SB to denote the safe belief semantics.

Theorem 28. *Let P be a propositional logic program, let F be a formula such that $\mathcal{L}_F \subseteq \mathcal{L}_P$ and let x be an atom not in $\mathcal{L}_P$. $P \xrightarrow{\text{SB}} P \cup \{x \leftrightarrow F\}$.*

Proof. Let M be a safe belief of $P \cup \{x \leftrightarrow F\}$ and let $M' = M \cap \mathcal{L}_P = M \setminus \{x\}$. Following Lemma 23, we obtain that $P \cup \{x \leftrightarrow F\} \cup \neg\neg E \cup \neg\neg X$ is a consistent and I-complete theory, where $E = M' \cup \neg(\mathcal{L}_P \setminus M')$ and the set X is either $\{x\}$ (if $x \in M$) or $\{\neg x\}$ (otherwise). Observe that the theory E is complete so that $E \vdash_{\mathrm{I}} F$ (or $\neg F$) where, because of consistency, the number of negations should match the number of negations of the atom x in X. It follows that $\{x \leftrightarrow F\} \cup \neg\neg E \vdash_{\mathrm{I}} \neg\neg X$ and thus we may remove $\neg\neg X$ from the list of premises to obtain the consistent and I-complete theory $P \cup \{x \leftrightarrow F\} \cup \neg\neg E$. In this statement the atom x now only appears in the premise $x \leftrightarrow F$ and therefore, by replacing x with F, we obtain the theorem $F \leftrightarrow F$ that we can also drop to finally obtain $P \cup \neg\neg E$ as a consistent and I-complete theory. It follows that E and M' correspond, respectively, to an I-explanation and a safe belief of P.

For the converse take a safe belief M of the logic program P and let E be any I-explanation of the program P such that $M = Cn_{\mathrm{I}}(P \cup \neg\neg E) \cap \mathcal{L}_P$. Since the theory $P \cup \neg\neg E$ is I-complete it decides, in particular, F. It follows that the theory $P \cup \{x \leftrightarrow F\} \cup \neg\neg E$ decides the new atom x and, therefore, it constitutes also an I-complete theory. Then we are able to conclude that the corresponding set $M' = Cn_{\mathrm{I}}(P \cup \{x \leftrightarrow F\} \cup \neg\neg E) \cap \mathcal{L}_{P \cup \{x \leftrightarrow F\}}$ is a safe belief of P. □

Conservative transformations may seem very effective in the context of logic programming. However, they dot not satisfy an important property for concrete programming applications. We would expect that making a conservative transformation of one piece of a program would also result, "globally", in a conservative transformation for the whole program.

This is not true for simple conservative transformations as just defined. Consider the two programs $P_1 = \{a \leftarrow \neg b\}$ and $P_2 = \{a,\ b \leftarrow b\}$. According to Definition 27, P_2 is a conservative transformation of P_1 in the safe belief semantics since they both have $\{a\}$ as their unique safe belief. However, replacing P_1 with P_2 in the larger program $P = \{a \leftarrow \neg b,\ b\}$, to obtain the program $P' = \{a,\ b \leftarrow b,\ b\}$, will break this relation. Now P has one safe belief $\{b\}$, while the program P' has only the safe belief $\{a, b\}$.

In order to ensure that making local transformations of code inside logic programs will preserve global equivalence, we introduce the notion of a strong conservative transformation.

Definition 29. Let Sem be a semantic operator for a class of programs C. Given two logic programs P and $P' \in C$, such that $\mathcal{L}_P \subseteq \mathcal{L}_{P'}$, we say that P' is a *strong conservative translation* of P, denoted $P \xrightarrow{\text{Sem}^*} P'$, if for every logic program Q, such that $\mathcal{L}_Q \cap (\mathcal{L}_{P'} \setminus \mathcal{L}_P) = \emptyset$, $P \cup Q \xrightarrow{\text{Sem}} P' \cup Q$.

The condition $\mathcal{L}_Q \cap (\mathcal{L}_{P'} \setminus \mathcal{L}_P) = \emptyset$ states that the programs Q, used to extend P, should not contain any of the reserved atoms of the transformation. In an actual implementation we could ensure this condition by defining a special set of atoms reserved for internal transformations and not available to the user for writing programs. As we may expect, strong conservative translations also define a transitive relation.

The particular case where there are no reserved atoms, i.e. $\mathcal{L}_P = \mathcal{L}_{P'}$, is known as strong equivalence between programs. This notion was originally introduced in [8] where the authors provide a characterization of strong equivalence for augmented programs, under the answer set semantics, using the HT logic. Their proof, however, is based on the equilibrium logic and thus can be immediately extended, following Proposition 9 and Theorem 25, to safe beliefs for arbitrary propositional theories.

A study of relations between answer sets and the logic G_3 is also presented in [13] where an alternative proof for the characterization of strong equivalence, that does not require the use of equilibrium models, is given.

Theorem 30 ([8,13]). *Let P and P' be two propositional logic programs such that $\mathcal{L}_P = \mathcal{L}_{P'}$. $P \xrightarrow{\text{SB}^*} P'$ if and only if $P \equiv_{G_3} P'$.*

An important feature of this result is that the corresponding proof also provides, if two programs P and P' are not strongly equivalent, a method to construct a certificate program Q, i.e. a program such that $P \cup Q$ and $P' \cup Q$ have different safe beliefs.

Example 31. Let $P = \{a \leftarrow \neg b\}$ and let $P' = \{a,\ b \leftarrow b\}$. Both programs are equivalent, in a weak sense, since they both have only one safe belief $\{a\}$. The proof method of Theorem 30, see for instance [13], can be used to construct the program $Q = \{\neg a, b\}$ such that $P \cup Q$ has one answer set, namely $\{b\}$, while the program $P' \cup Q$ is inconsistent and has no answer sets. Then we are able to conclude that the programs P and P' are not strongly equivalent.

5.2. *Polynomial reduction of theories*

A *translation* is a function $\text{Tr}: C \to C'$, where C and C' are two classes of logic programs. Janhunen [7] discusses important properties of program translations, relevant to logic programming, for arbitrary semantic operators. Particular applications for the answer set semantics are also given in [23].

Definition 32 (*[7,23]*). Let Sem be a semantic operator for a class of logic programs D, a translation $\mathrm{Tr}\colon C \to C'$, where the classes $C, C' \subseteq D$ are closed under unions,[6] is said to be:

polynomial if the time required to compute $\mathrm{Tr}(P)$ is polynomial with respect to the number of atomic formulas in P.
faithful if, for all programs $P \in C$, $P \xrightarrow{\mathrm{Sem}} \mathrm{Tr}(P)$.
strongly faithful if, for all programs $P \in C$, $P \xrightarrow{\mathrm{Sem}^*} \mathrm{Tr}(P)$.
modular if, for all programs $P_1, P_2 \in C$, $\mathrm{Tr}(P_1 \cup P_2) = \mathrm{Tr}(P_1) \cup \mathrm{Tr}(P_2)$.
reductive if $C' \subseteq C$ and $\mathrm{Tr}(P') = P'$ for all programs $P' \in C'$.[7]

The property of a translation being polynomial (P) is related with the complexity that an actual computer implementation of the translation should have. A faithful (F) translation can be applied to a program preserving the semantics, while a strongly faithful (S) translation can also be applied *locally* to some section of the program without altering the semantics.

The last two properties deal with the form of the translation, not with its particular semantics. If a translation is modular (M) we could split a program into several pieces and then perform the translation *piece by piece*. A reductive (R) translation maps one class of programs into some given subclass and the programs that already are in the "simplified" subclass are not modified by the translation.

As a form of notation we say that a translation is PFM if it is simultaneously polynomial, faithful and modular. Analogously, a PSMR translation is polynomial, strongly faithful, modular and reductive. We can drop any of the letters from the notation if we are just interested in some properties.

Proposition 33 (*[23]*). *There is a PSM translation, for the the semantics of safe beliefs,* $\mathrm{AugDis}\colon \mathsf{Aug} \to \mathsf{Dis}$.

Using the machinery of logic, based on Definition 8 and results like Theorem 30, it is also possible to provide a translation of logic programs, containing arbitrary propositional formulas, into augmented programs.

Definition 34. If a formula F contains a proper subformula $A \to B$, where neither A nor B contain more implications, we say that $A \to B$ is a *simple embedded implication* of the formula F. We define recursively the translation $\mathrm{PrpAug}\colon \mathsf{Prp} \to \mathsf{Aug}$ for every propositional program P, as follows:

- if P contains no clause with embedded implications then P is already an augmented program and $\mathrm{PrpAug}(P) = P$.
- if P contains some clause F with embedded implications take a simple embedded implication, $A \to B$, from the formula F. Take a new atom $x \in \mathcal{L} \backslash \mathcal{L}_P$ not already in P,

[6] A class of programs C is *closed under unions* if $P_1, P_2 \in C$ implies that $P_1 \cup P_2 \in C$.
[7] The definition of a *modular* translation we introduce here corresponds to the one given in [23]. Janhunen [7] presents a different definition which corresponds to *modular* + *reductive* (when $C' \subseteq C$ is satisfied).

let F' be the formula obtained by replacing the occurrence of $A \to B$ in F with the new atom x, and let P' be the program obtained by replacing F with F' in P. Also let

$$\Delta = \{x \wedge A \to B,\ \neg A \vee B \to x,\ x \vee A \vee \neg B\}\,.$$

We finally define $\text{PrpAug}(P) = \text{PrpAug}(P' \cup \Delta)$.

The recursive definition of PrpAug is well-founded since, on each recursion step, the program $P' \cup \Delta$ has one implication less than P.

Proposition 35. *The translation* PrpAug: $\mathsf{Prp} \to \mathsf{Aug}$ *is PSMR.*

Proof. The number of recursion steps required to complete the translation is exactly the number of embedded implications in the program, therefore the translation is polynomial. It is also clear, since PrpAug acts on one clause at a time and does not modify programs already in the augmented class, that the translation is modular and reductive.

To justify that the recursion step is a strong conservative transformation observe that, from Theorem 28, $P \xrightarrow{\text{SB}} P \cup \{x \leftrightarrow (A \to B)\}$. The key point is that $\{x \leftrightarrow (A \to B)\} \equiv_{\text{G}_3} \Delta$ and, therefore, we also have the equivalence $P \cup \{x \leftrightarrow (A \to B)\} \equiv_{\text{G}_3} P' \cup \Delta$. Using Theorem 30, and transitivity, we end up with $P \xrightarrow{\text{SB}} P' \cup \Delta$. $\square$

Current implementations of the answer set programming paradigm restrict the syntax of formulas to the class of disjunctive programs where, in particular, implication in the body is not allowed. A common work around this limitation was to use the intuitive (classical) equivalence $(A \to B) \leftrightarrow (\neg A \vee B)$. This practice, however, sometimes has the effect of generating unexpected models (or the miss of expected ones) when computing answer sets.

The first rule $x \wedge A \to B$ in our equivalence is used to model the behavior of the implication symbol in the head. The second rule $\neg A \vee B \to x$ comes from the classical intuitive meaning of the implication connective. These two rules, however, are not enough to provide the required equivalence in the logic G_3. The less intuitive third rule $x \vee A \vee \neg B$ — required for the equivalence to hold — was discovered, in fact, by an examination of the G_3 models of the original formula $x \leftrightarrow (A \to B)$. This points out the importance of results like Theorem 30 that allows us to better understand the notion of answer sets, proposing the logic G_3 as a more correct guide for our intuition.

The results presented in this section allow us to reduce arbitrary propositional theories to the simple class of disjunctive logic programs. Thus providing a first approach to compute safe beliefs of a logic program.

Theorem 36. *There is a PSM translation* PrpDis: $\mathsf{Prp} \to \mathsf{Dis}$.

Proof. It follows immediately after Propositions 33 and 35. $\square$

6. Conclusions

We provide a general approach, that we call safe beliefs, that can be used to study several properties and concepts of the answer set semantics. This approach is based on extensions of theories obtained by adding double negated formulas to logic programs in order to obtain consistent and complete explanations. An important virtue of this formulation is that it can be easily extended for other logics to produce different (sometimes non-monotonic) inference systems.

One of the main contributions of this paper is to show that, in particular, any proper intermediate logic can be used instead of intuitionistic logic to define safe beliefs and obtain the same semantic operator that extends answer sets for propositional theories. As an important consequence we obtain that the alternate extension of answer sets provided by the equilibrium logic is equivalent to our safe beliefs semantics. We were able then to obtain a lot of feedback between the two different approaches.

We also study properties of translations for logic programs in the context of answer set programming. As another contribution we show that logic programs can be extended, introducing new atoms by definition, without modifying the semantical properties of the program. This seems a desirable property that should be taken into account when studying other semantics.

Finally we provide a translation that can be used to reduce arbitrary propositional formulas to the class of augmented logic programs. Other reductions can be used to complete the reductions and reach the class of disjunctive logic programs. These sorts of translations are important, since they offer a natural approach to solve the problem of computing safe beliefs for arbitrary propositional theories. Reductions, like those presented in Section 4.2, can also be useful to build efficient algorithms to compute answer sets.

An interesting topic for further research is to study possible generalizations and extensions of the safe beliefs idea to incorporate linear or modal logics. This could provide a logic programming paradigm for environments with limited resources, or where several agents can solve problems reasoning about the knowledge and beliefs of each other.

Acknowledgement

Numerous discussions with David Pearce helped us to identify the relations between the safe beliefs approach and the equilibrium logic.

References

[1] J.C. Acosta Guadarrama, J. Arrazola, M. Osorio, Making belief revision with LUPS, in: J.H.S. Azuela, G.A. Figueroa (Eds.), XI International Conference on Computing, México, D.F., November 2002, CIC-IPN, 2002.

[2] C. Baral, Knowledge Representation, Reasoning and Declarative Problem Solving with Answer Sets, Cambridge University Press, Cambridge, 2003.

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

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

[5] K. Eshgi, R. Kowalski, Abduction compared with negation by failure, in: G. Levi, M. Martelli (Eds.), Logic Programming, Proceedings of the Sixth International Conference, June 1989, Lisbon, Portugal, MIT Press, 1989, pp. 234–255.

[6] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R. Kowalski, K. Bowen (Eds.), 5th Conference on Logic Programming, MIT Press, 1988, pp. 1070–1080.
[7] T. Janhunen, On the effect of default negation on the expressiveness of disjunctive rules, in: T. Eiter, W. Faber, M. Truszczynski (Eds.), Logic Programming and Nonmonotonic Reasoning, 6th International Conference, September 2001, Vienna, Austria, Lecture Notes in Computer Science, vol. 2173, Springer, 2001, pp. 93–106.
[8] V. Lifschitz, D. Pearce, A. Valverde, Strongly equivalent logic programs, ACM Transactions on Computational Logic 2 (2001) 526–541.
[9] V. Lifschitz, G. Schwarz, Extended logic programs as autoepistemic theories, in: L.M. Pereira, A. Nerode (Eds.), 2nd International Workshop on Logic Programming & Non-Monotonic Reasoning, Cambridge, MA, MIT Press, 1993, pp. 101–114.
[10] V. Lifschitz, L.R. Tang, H. Turner, Nested expressions in logic programs, Annals of Mathematics and Artificial Intelligence 25 (1999) 369–389.
[11] V.W. Marek, M. Truszczyński, Reflexive autoepistemic logic and logic programming, in: L.M. Pereira, A. Nerode (Eds.), 2nd International Workshop on Logic Programming & Non-Monotonic Reasoning, Cambridge, MA, MIT Press, 1993, pp. 115–131.
[12] E. Mendelson, Introduction to Mathematical Logic, third ed., Wadsworth, Belmont, CA, 1987.
[13] J.A. Navarro, 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, August, 2002.
[14] J.A. Navarro, Properties of translations for logic programs, in: Balder ten Cate (Ed.), Eight ESSLLI Student Session, European Summer School in Logic, Language and Information, Vienna, Austria, August, 2003.
[15] I. Niemelä, P. Simons, 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, September 1996, Bonn, Germany, The MIT Press, 1996, pp. 289–303.
[16] M. Osorio, J.A. Navarro, J. Arrazola, Equivalence in answer set programming, in: A. Pettorossi (Ed.), Logic Based Program Synthesis and Transformation, 11th International Workshop, LOPSTR 2001, November, Paphos, Cyprus, LNCS, vol. 2372, Springer, 2001, pp. 57–75.
[17] M. Osorio, J.A. Navarro, J. Arrazola, Debugging in A-Prolog: A logical approach (abstract), in: P.J. Stuckey (Ed.), Logic Programming. 18th International Conference, ICLP 2002, August 2002, Copenhagen, Denmark, LNCS, vol. 2401, Springer, 2002, pp. 482–483.
[18] M. Osorio, J.A. Navarro, J. Arrazola, A logical approach for A-Prolog, in: R. de Queiroz, L.C. Pereira, E.H. Haeusler (Eds.), 9th Workshop on Logic, Language, Information and Computation, WoLLIC, Rio de Janeiro, Brazil, Electronic Notes in Theoretical Computer Science, vol. 67, Elsevier Science Publishers, 2002, pp. 265–275.
[19] M. Osorio, J.A. Navarro, J. Arrazola, Applications of intuitionistic logic in answer set programming, Theory and Practice of Logic Programming 4 (2004) 325–354.
[20] M. Osorio, J.C. Nieves, C. Giannella, Useful transformations in answer set programming, in: A. Provetti, T.C. Son (Eds.), Proceedings of the American Association for Artificial Intelligence, AAAI 2001 Spring Symposium Series, Stanford, E.U., AAAI Press, 2001, pp. 146–152.
[21] D. Pearce, From here to there: stable negation in logic programming, in: D.M. Gabbay, H. Wansing (Eds.), What is Negation?, Kluwer Academic Publishers, Netherlands, 1999, pp. 161–181.
[22] D. Pearce, Stable inference as intuitionistic validity, Logic Programming 38 (1999) 79–91.
[23] D. Pearce, V. Sarsakov, T. Schaub, H. Tompits, S. Woltran, 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, August 2002, Copenhagen, Denmark, LNCS, vol. 2401, Springer, 2002, pp. 405–420.
[24] A.S. Troelstra, D. van Dalen, Constructivism in Mathematics: An Introduction, vol. II, North-Holland, Amsterdam, 1988.
[25] D. van Dalen, Logic and Structure, second ed., Springer, Berlin, 1980.
[26] M. Zakharyaschev, F. Wolter, A. Chagrov, Advanced modal logic, in: D.M. Gabbay, F. Guenthner (Eds.), Handbook of Philosophical Logic, second ed., vol. 3, Kluwer Academic Publishers, Dordrecht, 2001, pp. 83–266.