Properties of Translations for Logic Programs

Juan Antonio Navarro Pérez
Universidad de las Américas - Puebla
ma108907@mail.udlap.mx

Abstract. We present a study of some properties of program translations in the context of logic programming. In particular we provide, under the answer set semantics, a translation for arbitrary propositional theories into the simple class of disjunctive programs. We also show how syntactic and semantic properties of translations can be related. The work follows a line of research that applies mathematical logic to study notions and concepts in logic programming.

1 Introduction

Program translations are functions that map logic programs in a given class of programs to another class. Translations for logic programs can be very interesting for several different reasons: they can allow to simplify the structure of programs (Osorio et al. 2001; Pearce et al. 2002; Sakama and Inoue 1998), to derive correct programs from specifications (Pettorossi and Proietti 1998), and even to perform program updates and belief revision for agent systems (Alferes et al. 2002; Eiter et al. 2000).

We are interested in the study of the properties that translations should have in order to preserve the semantical meaning of programs. In particular we want to investigate this sort of properties in the context of Answer Set Programming (ASP). This semantic, originally introduced by Gelfond and Lifschitz (1988) and generalized later to include broader classes of programs (Lifschitz et al. 1999; Osorio et al. 2003), rapidly became a popular logic programming paradigm.

The great power of this semantic to express a wide variety of problems, and the existence of very efficient software to compute answer sets, allowed the development of several real life applications based on ASP. The possibilities range from solving combinatorial problems, modeling logic agents, planning (Dimopoulos et al. 1997), knowledge representation (Baral 2003) and querying deductive databases (Eiter et al. 1997); just to mention a few.

Proceedings of the Eighth ESSLLI Student Session
Balder ten Cate (editor)

Previous work from Janhunen (2001) studies several syntactic and semantic properties of translations for arbitrary semantic operators. This kind of properties are relevant for several theoretical and practical reasons, as we will discus along this presentation. Pearce et al. (2002) shows, in particular, a translation for programs containing nested expressions to the class of disjunctive programs that preserves the original semantics of programs.

In this paper we follow an approach, motivated from (Pearce 1999; Osorio et al. 2003), that tries to explain answer set programming in terms of intermediate logics. We provide a translation, with nice syntactical and semantical properties, that can reduce propositional theories to the class of nested programs considered by Pearce et al. (2002).

We also show that, under certain simple assumptions, the syntactic properties of a translation that preserves the answer set semantics can be enough to be sure that the translation can also preserve the semantics for partially translated programs. This sort of results show how the use of intuitionistic and other intermediate logics is an approach that can help us to better understand the notion and concepts of answer sets.

Because of the lack of space, the proofs of some minor results were left out from the main discussion of the paper. A complete version of the paper, including an appendix with all the omitted proofs, will be made available at `http://www.udlap.mx/~ma108907/papers.html`.

2 Background

We review in this section the language of propositional logic used to describe logic programs. We briefly introduce intuitionistic and some multivalued-logics, state some notation and basic results. A more detailed presentation can be found at (Mendelson 1987; Zakharyaschev et al. 2001). We also define several concepts for logic programming: classes of logic programs, semantic operators and answer sets. Original definitions and related results are available in (Lloyd 1987; Lifschitz et al. 1999; Osorio et al. 2003).

Propositional Logic We consider a formal language built from an alphabet containing: a denumerable set $\mathcal{L}$ of elements called atoms, 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 $A \leftrightarrow B$ to abbreviate 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 will restrict our attention, however, to finite theories. We use Prp to denote the set of all finite propositional theories. For a given theory T its *signature* is the set $\mathcal{L}_T$ of atoms occurring in the theory T. A *literal* is either an atom a or a negated atom $\neg a$. Given a theory T we also define the negated theory $\neg T = \{\neg A \mid A \in T\}$ and, for a set of atoms M, the *closure* of M (w.r.t. T) as $\overline{M} = M \cup \neg(\mathcal{L}_T \setminus M)$.

Intermediate Logics Intuitionistic logic, denoted I, was developed as an alternative to classical logic. It explains the meaning of the connectives in terms of *knowledge* or *provability*, instead of absolute *truths.* We consider a formulation of intuitionistic logic where a *theorem* is a formula that can be proved using the ten axioms that define I and *modus ponens* as inference rule (van Dalen 1980).

Gödel also 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. The two valued logic G_2 corresponds to the classical logic, denoted C. (Mendelson 1987)

It was shown that the set of intuitionistic theorems is a proper subset of the set of classical tautologies, and that the logics G_i lie in between. So we call *intermediate logics* to those logics whose set of provable formulas is between intuitionistic and classical logics (inclusive). A *proper intermediate logic* is an intermediate logic that is not the classical one.

Notation and Basic Results We use the notation $\vdash_{\mathrm{X}} F$ to denote that the formula F is provable (a theorem or tautology) in logic X. If T is a theory we use the symbol $T \vdash_{\mathrm{X}} F$ to denote $\vdash_{\mathrm{X}} (F_1 \wedge \cdots \wedge F_n) \to F$ for some formulas $F_i \in T$. We say that a theory T is *consistent* if it is not the case that $T \vdash_{\mathrm{C}} \bot$. It is easy to prove that the definition of a theory of being consistent does not depend on the logic chosen. The proof is given in the appendix at the end of the full version of this document.

We also introduce, if T and U are two theories, the symbol $T \vdash_{\mathrm{X}} U$ to denote that $T \vdash_{\mathrm{X}} F$ for all formulas $F \in U$. We will write $T \Vdash_{\mathrm{X}} U$ to denote the fact that (i) T is consistent and (ii) $T \vdash_{\mathrm{X}} U$. Finally we say that two theories T_1 and T_2 are equivalent (w.r.t. logic X), denoted $T_1 \equiv_{\mathrm{X}} T_2$ if it holds that both $T_1 \vdash_{\mathrm{X}} T_2$ and $T_2 \vdash_{\mathrm{X}} T_1$.

Logic Programs In order to introduce the terminology of logic programs, using the propositional logic we have just presented, we define a *clause* as a formula $H \leftarrow B$ where principal connective is implication. The formulas H and B are known as the *head* and the *body* of the clause respectively.

The special case of a clause with the form $\bot \leftarrow B$ is known as a *constraint*, in this case the head is said to be *empty.* Each formula H that does not have implication as the principal connective can be associated with the clause $H \leftarrow \top$, this kind of clauses are known as *facts.* Then we can define a *logic program* as a finite set of clauses and a *class of logic programs* as some set of logic programs.

There are several kind of clauses defined in literature. The head and the body of *augmented clauses* are constructed from the connectives $\wedge$, $\vee$ and $\neg$ arbitrarily nested. Note that the general case of implication is not allowed. The clause $\neg(a \wedge \neg b) \leftarrow p \vee (\neg q \wedge r)$ is, for example, augmented while the clause $a \leftarrow (b \rightarrow c)$ is not. An *augmented program* is a set of augmented clauses and Aug denotes the class of all augmented programs.

Another, more restricted, class is the disjunctive one. A *disjunctive clause* allows only a (non-empty) disjunction of positive literals in the head and a conjunction of literals in the body, for example $a \vee b \leftarrow p \wedge q \wedge \neg r$. Similarly, a *disjunctive program* is a set of disjunctive clauses and Dis is the class of all disjunctive programs. Observe that we have $\mathsf{Dis} \subset \mathsf{Aug} \subset \mathsf{Prp}$.

Answer Sets Given a class of programs C, a *semantic operator* Sem is a function that assigns to each program $P \in C$ a set of sets of atoms $M \subseteq \mathcal{L}_P$. These sets of atoms are usually some "preferred" models of the program P. One popular semantic operator is the answer sets AS operator. We consider the definition provided in (Osorio, Navarro, and Arrazola 2002) that extends the notion of an answer set to the whole set of propositional formulas.

Definition 1. If $P \in \mathsf{Prp}$ then $\mathrm{AS}(P) = \{M \subseteq \mathcal{L}_P \mid P \cup \neg\neg\overline{M} \Vdash_{\mathrm{I}} M\}$.[1]

3 Conservative Transformations

Suppose that we have a logic program P and we want to compute $\mathrm{AS}(P)$. We could simplify this task if we are able to construct some other simpler program P', such that the answer sets of P and P' are somehow related, compute $\mathrm{AS}(P')$ and recover the answer sets of the original P. It will be important that this "recovery" of the answer sets of P, knowing those of P', can be done trough a simple and efficient method. A conservative transformation is a relation between logic programs with this kind of properties.

Definition 2. Let Sem be a semantic operator defined for a class of programs C and let $P, P' \in C$. We say P' is a *conservative transformation* of P, denoted $P \xrightarrow{\mathrm{Sem}} P'$, if $\mathcal{L}_P \subseteq \mathcal{L}_{P'}$ and

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

A conservative extension, as presented in (Baral 2003), is a conservative transformation where the programs also satisfy $P \subseteq P'$. Our definition is more general since it allows the program to be modified or "transformed" and not only extended. As far as we know this definition is new, and so the results presented in this section are all original.

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 this newly introduced atoms, we obtain exactly the answer sets of P. We refer to this new atoms in $\mathcal{L}_{P'} \setminus \mathcal{L}_P$ as the *reserved* atoms of the transformation.

[1]The original definition uses the notation $\neg M \cup \neg\neg\widetilde{M}$, where $\widetilde{M} = \mathcal{L}_P \setminus M$, instead of the set $\neg\neg\overline{M}$, where $\overline{M} = M \cup \neg(\mathcal{L}_P \setminus M)$. It is easy to verify that the two conditions are equivalent since $\vdash_{\mathrm{I}} \neg a \leftrightarrow \neg\neg\neg a$.

It is easy to verify that conservative transformations define a transitive relation, a formal proof is included in the appendix. Much more surprising is that arrows in the notation of conservative extensions can sometimes be traveled backwards. If a logic program Q is a conservative extension of two programs P and R then, under the premise that $\mathcal{L}_P \subseteq \mathcal{L}_R$, we can also relate the answer sets of P and R by a conservative transformation.

Proposition 1. *If* $P \xrightarrow{\text{Sem}} Q$, $R \xrightarrow{\text{Sem}} Q$, *and* $\mathcal{L}_P \subseteq \mathcal{L}_R$ *then* $P \xrightarrow{\text{Sem}} R$.

Proof. Take $M \in \text{Sem}(R)$, since $R \xrightarrow{\text{Sem}} Q$ there must be $N \in \text{Sem}(Q)$ such that $M = N \cap \mathcal{L}_R$. But, since $P \xrightarrow{\text{Sem}} Q$, we have that $N \cap \mathcal{L}_P \in \text{Sem}(P)$. Observe, since $\mathcal{L}_P \subseteq \mathcal{L}_R$, that $M \cap \mathcal{L}_P = (N \cap \mathcal{L}_R) \cap \mathcal{L}_P = N \cap \mathcal{L}_P$. Thus we obtain $M \cap \mathcal{L}_P \in \text{Sem}(P)$.

Now take $M \in \text{Sem}(P)$, since $P \xrightarrow{\text{Sem}} Q$ there must be $N \in \text{Sem}(Q)$ such that $M = N \cap \mathcal{L}_P$. But, since $R \xrightarrow{\text{Sem}} Q$, we have that $N \cap \mathcal{L}_R \in \text{Sem}(R)$. Observe, since $\mathcal{L}_P \subseteq \mathcal{L}_R$, that $(N \cap \mathcal{L}_R) \cap \mathcal{L}_P = N \cap \mathcal{L}_P = M$. Then $N \cap \mathcal{L}_R \in \text{Sem}(R)$ is the model such that $(N \cap \mathcal{L}_R) \cap \mathcal{L}_P \in \text{Sem}(P)$. □

The following proposition allows us to construct a very simple conservative translation for the semantic of answer sets. It can be used to extend the language of a given program without modifying its answer sets. This result will be used later in the proof of Theorem 3.

Proposition 2. *Given a set of atoms* A, *let* $L = \{a \leftarrow a \mid a \in A\}$. *For any program* $P \in \mathsf{Prp}$, $P \xrightarrow{\text{AS}} P \cup L$.

Another particular case of a conservative transformation, which is presented in the following proposition, stands that is possible to introduce new atoms as a definition of a formula that can be expressed using the atoms already in the program.

Proposition 3. *Let* $P \in \mathsf{Prp}$. *Given a formula* F *such that* $\mathcal{L}_F \subseteq \mathcal{L}_P$ *and an atom* $x \notin \mathcal{L}_P$, $P \xrightarrow{\text{AS}} P \cup \{x \leftrightarrow F\}$.

Conservative transformations may seem very effective in the context of logic programming. But they dot not satisfy, in general, 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 2, P_2 is a conservative translation of P_1 in the answer set semantics since they both have $\text{AS}(P_1) = \text{AS}(P_2) = \{\{a\}\}$. 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 only one answer set $\{b\}$, while P' has the answer set $\{a, b\}$.

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 3. Let Sem be a semantic operator for a class of programs C. Given two logic programs $P, P' \in C$, 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 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, may not contain any of this reserved atoms of the transformation. In an actual implementation we could ensure this condition by defining a special set of atoms reserved for internal translations and not available to the user for writing programs. As we may expect, strong conservative translations also define a transitive relation.

In the particular case when $\mathcal{L}'_P = \mathcal{L}_P$, when there are no reserved atoms, this relation is known as a strong equivalence between programs. This notion was originally introduced in (Lifschitz, Pearce, and Valverde 2001) where the authors provide a characterization of strong equivalence for augmented programs, using the answer set semantics, in terms of the logic HT equivalent to the 3 valued logic G_3. These relations between answer sets and intermediate logics are also studied in (Navarro 2002) where the characterization of strong equivalence is revised and extended to arbitrary propositional theories.

Theorem 1. *(Lifschitz et al. 2001; Navarro 2002) Let $P, P' \in \mathsf{Prp}$, such that $\mathcal{L}_P = \mathcal{L}_{P'}$. $P \xrightarrow{\text{AS}^*} P'$ if and only if $P \equiv_{\text{G}_3} P'$.*

4 Program Translations

A *translation* is a function $\text{Tr}\colon C \to C'$, where C and C' are two classes of logic programs. Janhunen (2001) 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 (Pearce et al. 2002).

Definition 4. (Janhunen 2001; Pearce et al. 2002) Let Sem be a semantic operator for a class of programs D, a translation $\text{Tr}\colon C \to C'$, where the classes $C, C' \subseteq D$ are closed under unions[2], is said to be:

polynomial if the time required to compute $\text{Tr}(P)$ is polynomial with respect to the number of symbols in P.

[2]A class of programs C is *closed under unions* if $P_1, P_2 \in C$ implies that $P_1 \cup P_2 \in C$.

faithful if, for all programs $P \in C$, $P \xrightarrow{\text{Sem}} \mathrm{Tr}(P)$.

strongly faithful if, for all programs $P \in C$, $P \xrightarrow{\text{Sem}^*} \mathrm{Tr}(P)$.

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 $P' \in C'$.[3]

The property of a translation being polynomial (P) is related with the order of 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 are already in the subclass will not been 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 4. *(Pearce et al. 2002) There is a PSM translation, for the the semantic of answer sets,* $\mathrm{AugDis}\colon \mathsf{Aug} \to \mathsf{Dis}$.

Using the machinery of logic, based on Definition 1 and results like Theorem 1, it is also possible to provide a translation of logic programs, containing arbitrary propositional formulas, into augmented programs.

Definition 5. If a formula F contains a proper subformula $A \to B$, where A and B contain no 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 $P \in \mathsf{Prp}$, as follows:

(i) P contains no clause with embedded implications. Then P is already an augmented program and $\mathrm{PrpAug}(P) = P$.

(ii) P contains a clause F with some embedded implications. Take one simple embedded implication, $A \to B$, from the formula F; and take a new atom $x \in \mathcal{L} \setminus \mathcal{L}_P$ not already in P. Let F' be the formula obtained

[3]The definition of a *modular* translation we introduce here corresponds to the one given in (Pearce et al. 2002). Janhunen (2001) presents a different definition which corresponds to *modular* + *reductive* (when $C' \subseteq C$ is satisfied).

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 $\mathrm{PrpAug}(P) = \mathrm{PrpAug}(P' \cup \Delta)$.

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 5. *The translation* $\mathrm{PrpAug}\colon \mathsf{Prp} \to \mathsf{Aug}$ *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 Proposition 3, $P \xrightarrow{\mathrm{AS}} P \cup \{x \leftrightarrow (A \to B)\}$. The key point is that $\{x \leftrightarrow (A \to B)\} \equiv_{\mathrm{G}_3} \Delta$ and, therefore, we also have the equivalence $P \cup \{x \leftrightarrow (A \to B)\} \equiv_{\mathrm{G}_3} P' \cup \Delta$. Using Theorem 1, and transitivity, we end up with $P \xrightarrow{\mathrm{AS}} P' \cup \Delta$. □

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 to this limitation was to use the intuitive equivalence $(A \to B) \leftrightarrow (\neg A \vee B)$. This practice, however, caused sometimes the appearance of 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. This two rules, however, are not enough to provide the required equivalence in the logic G_3. This could possibly explain the unexpected results obtained from the erroneous work around. The less intuitive third rule $x \vee A \vee \neg B$ required 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 1 that allows us to better understand the notion of answer sets, proposing the logic G_3 as a more correct guide for our intuition.

5 Properties of Translations

In the previous section we provided a translation for the class of propositional theories into the class of augmented programs and, together with other translations (Pearce et al. 2002), it is possible to reduce them to the class of disjunctive programs. Using any of the popular answer set finders for disjunctive programs, `dlv`[4] or `smodels`[5], it is possible to provide an efficient method to compute answer sets for propositional theories. The following theorem is a direct consequence from Propositions 4 and 5.

Theorem 2. *There is a translation* PrpDis: $\mathsf{Prp} \to \mathsf{Dis}$*, which is PSM for the semantic of answer sets.*

Observe that the properties of strongly faithful and modular are somehow related. Both notions can be interpreted in terms of a program that has been split into several pieces. A strongly faithful translation can be applied to one of these pieces preserving the semantics of the program. On the other hand, a modular translation can also be applied "piece by piece" to the program but we do have to complete the translation for each one of the pieces. It is possible, indeed, to construct a FM translation which is not strongly faithful.

Example 1. Let C be the class of disjunctive programs that have exactly one atom in the head and zero or one atoms in the body. Also let C' be the class of disjunctive programs that have exactly two atoms in the head and zero or two atoms in the body. Clearly both C and C' are closed under unions. For each atom a in the user language let a' be a new reserved atom. The translation Hide: $C \to C'$ is defined mapping each clause, a or $a \leftarrow b$, as follows:

$$\begin{array}{lcl}\mathrm{Hide}(\{a\}) & = & \{a' \vee a',\, a \vee a \leftarrow a' \wedge a'\} \\ \mathrm{Hide}(\{a \leftarrow b\}) & = & \{a' \vee a' \leftarrow b' \wedge b',\, a \vee a \leftarrow a' \wedge a',\, b \vee b \leftarrow b' \wedge b'\}\end{array}$$

It is clear, by construction, that the translation is modular. The translation is also faithful since it rewrites the original program, with new reserved atoms, and appending a set of rules, equivalent to $a \leftarrow a'$, in order to recover answer sets in the original signature.

Consider the program $P = \{a \leftarrow b,\, b\} \in C$. Both P and Hide(P) have the same answer sets. If we apply the translation, however, just to the first clause the program $\mathrm{Hide}(\{a \leftarrow b\}) \cup \{b\}$ will have different semantics. The rule $a \leftarrow b$ is now "hidden" and the existence of the fact b can not be used anymore to infer a. The translation is not strongly faithful.

We will see, however, that there is a wide class of interesting and useful translations where the syntactic properties of being modular and reductive are enough to be strongly faithful. The following theorem shows how, in the context of the answer sets semantics, this can be possible.

Theorem 3. *Given two classes of logic programs C and C', closed under unions and such that* $\mathsf{Dis} \subseteq C' \subseteq C \subseteq \mathsf{Prp}$*. If the translation* $\mathrm{Tr}\colon C \to C'$ *is FMR in the answer set semantics then it is also strongly faithful.*

[4]`http://www.dbai.tuwien.ac.at/proj/dlv/`

[5]`http://www.tcs.hut.fi/Software/smodels/`

Proof. Let $Q \in \mathsf{Prp}$ be a program containing no atoms from $\mathcal{L}_{\mathrm{Tr}(P)} \setminus \mathcal{L}_P$ and let $L = \{a \leftarrow a \mid a \in \mathcal{L}_{\mathrm{Tr}(P)}\}$, so that the signatures are $\mathcal{L}_P \subseteq \mathcal{L}_{\mathrm{Tr}(P)} = \mathcal{L}_L$. Construct then the disjunctive program $D = \mathrm{PrpDis}(Q \cup L)$ and, since Theorem 2 states that the translation is strongly faithful, $Q \cup L \xrightarrow{\mathrm{AS}^*} D$.

Neither P nor $\mathrm{Tr}(P)$ contain reserved atoms from $\mathcal{L}_D \setminus \mathcal{L}_{Q \cup L}$. Using the result from Proposition 2 and from the definition of strongly faithful we obtain that $P \cup Q \xrightarrow{\mathrm{AS}} P \cup (Q \cup L) \xrightarrow{\mathrm{AS}} P \cup D$ and, similarly for the translated program, $\mathrm{Tr}(P) \cup Q \xrightarrow{\mathrm{AS}} \mathrm{Tr}(P) \cup (Q \cup L) \xrightarrow{\mathrm{AS}} \mathrm{Tr}(P) \cup D$.

Now, since the translation Tr is faithful, we have $P \cup D \xrightarrow{\mathrm{AS}} \mathrm{Tr}(P \cup D)$. Also, from the modular and reductive properties, we obtain $\mathrm{Tr}(P \cup D) = \mathrm{Tr}(P) \cup \mathrm{Tr}(D) = \mathrm{Tr}(P) \cup D$. Thus we get $P \cup Q \xrightarrow{\mathrm{AS}} P \cup D \xrightarrow{\mathrm{AS}} \mathrm{Tr}(P) \cup D$ and, by Proposition 1, we can finally conclude $P \cup Q \xrightarrow{\mathrm{AS}} \mathrm{Tr}(P) \cup Q$. □

6 Conclusions

We have presented in this paper a translation that can be used to map arbitrary propositional theories into the class of simple disjunctive programs. Moreover, the translation is strongly faithful and has good syntactical properties. Previous work, from Pearce et al. (2002), presented a similar translation but only for programs in the augmented class. Using results from Lifschitz et al. (2001) and Osorio et al. (2003) we could show how to extend this translation for any propositional theory.

The existence of such translation has some theoretical significance. It shows, in particular, that the class of disjunctive programs is as expressive as the class of propositional theories. Also an important practical consequence of the result is that it allows the development of software tools to compute answer sets for logic programs containing arbitrary propositional formulas.

We also exhibit how, for a wide range of program translations, the conditions of being faithful, modular and reductive are sufficient to state that the translation is also strongly faithful. The main assumption required for this result to hold is that the translation takes programs from a class of logic programs to some, more simple in principle, subclass.

It is important to stress out the fact that this results were obtained through the "Logic Programming via Logic" approach initiated by Pearce (1999) and developed later by Osorio et al. (2003). Thus showing how the use of intermediate logics in the study of answer sets can be useful to develop the theory and applications of answer sets. This could open a new line of research and provide a lot of feedback between this two areas.

Acknowledgments This research is sponsored by the Mexican National Council of Science and Technology, CONACyT (project 37837-A).

Bibliography

Alferes, J. J., L. M. Pereira, H. Pryzmusinska, and T. Prymusinski (2002). LUPS—a language for updating logic programs. *Artificial Intelligence 138*, 87–116.

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

Dimopoulos, Y., B. Nebel, and J. Koehler (1997). Encoding planning problems in non-monotonic logic programs. In *Proceedings of the Fourth European Conference on Planning*, pp. 169–181. Springer-Verlag.

Eiter, T., M. Fink, G. Sabbatini, and H. Tompits (2000). On updates of logic programs: Semantics and properties. Research Report 1843-00-08, INFSYS. A shortened version of this paper appears in *Theory and Practice of Logic Programming*, 2002.

Eiter, T., N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello (1997, June). The architecture of a disjunctive deductive database system. In M. Falaschi, M. Navarro, and A. Policriti (Eds.), *Proceedings Joint Conference on Declarative Programming (APPIA-GULP-PRODE '97)*, pp. 141–151.

Gelfond, M. and V. Lifschitz (1988). The stable model semantics for logic programming. In R. Kowalski and K. Bowen (Eds.), *5th Conference on Logic Programming*, pp. 1070–1080. MIT Press.

Janhunen, T. (2001, September). 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*, Number 2173 in Lecture Notes in Computer Science, Vienna, Austria, pp. 93–106. Springer.

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

Lifschitz, V., L. R. Tang, and H. Turner (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.). Berlin: Springer.

Mendelson, E. (1987). *Introduction to Mathematical Logic* (Third ed.). Belmont, CA: Wadsworth.

Navarro, J. A. (2002, August). 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.

Osorio, M., J. A. Navarro, and J. Arrazola (2001, November). Equivalence in answer set programming. In A. Pettorossi (Ed.), *Logic Based Program Synthesis and Transformation. 11th International Workshop, LOPSTR 2001*, Number 2372 in LNCS, Paphos, Cyprus, pp. 57–75. Springer.

Osorio, M., J. A. Navarro, and J. Arrazola (2002). 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)*, Volume 67 of *Electronic Notes in Theoretical Computer Science*, Rio de Janeiro, Brazil, pp. 265–275. Elsevier Science Publishers.

Osorio, M., J. A. Navarro, and J. Arrazola (2003). Applications of intuitionistic logic in answer set programming. Accepted to appear at the TPLP journal.

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

Pearce, D., V. Sarsakov, T. Schaub, H. Tompits, and S. Woltran (2002, August). 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*, Number 2401 in LNCS, Copenhagen, Denmark, pp. 405–420. Springer.

Pettorossi, A. and M. Proietti (1998). Transformation of logic programs. In C. J. H. D. M. Gabbay and J. A. Robinson (Eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming*, Volume 5, Chapter D, pp. 697–787. Oxford University Press.

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

van Dalen, D. (1980). *Logic and Structure* (Second ed.). Berlin: Springer.

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

Appendix: Omitted Proofs

It is stated in the paper that the property of a program of being consistent does not depend on the logic underlying. This result is proved precisely in Proposition 6 and is based on the following well known results.

Lemma 1. *(Zakharyaschev et al. 2001) If the symbol $\subset$ denotes the proper set inclusion of the sets of provable formulas, intermediate logics are ordered as follows:* $\mathrm{I} \subset \cdots \subset \mathrm{G}_{i+1} \subset \mathrm{G}_i \subset \cdots \subset \mathrm{G}_3 \subset \mathrm{G}_2 = \mathrm{C}$.

Lemma 2. *(Mendelson 1987; Osorio et al. 2003) Given a theory T and a formula F, $T \vdash_{\mathrm{C}} F$ if and only if $T \vdash_{\mathrm{I}} \neg\neg F$.*

Proposition 6. *Let* X *and* Y *be two logics, selected among classical and intermediate logics. A theory P is consistent with respect to logic* X *iff it is consistent with respect to logic* Y*.*

Proof. It suffices to show that if P is inconsistent, with respect to X, then P is also inconsistent with respect to Y. Suppose that there is a theory P such that $P \vdash_{\mathrm{X}} \bot$ then, by Lemma 1, $P \vdash_{\mathrm{C}} \bot$. By Lemma 2 we get that $P \vdash_{\mathrm{I}} \neg\neg\bot$. But $\neg\neg\bot \to \bot$ is an intuitionistic theorem, so $P \vdash_{\mathrm{I}} \bot$. Finally, again by Proposition 1, we conclude that $P \vdash_{\mathrm{Y}} \bot$. $\square$

We present also the proofs that both conservative transformations and strong conservative transformations define transitive relations.

Proposition 7. *If $P \xrightarrow{\mathrm{Sem}} Q$ and $Q \xrightarrow{\mathrm{Sem}} R$ then $P \xrightarrow{\mathrm{Sem}} R$.*

Proof. Take $M \in \mathrm{Sem}(P)$ then, since $P \xrightarrow{\mathrm{Sem}} Q$, there is $M' \in \mathrm{Sem}(Q)$ such that $M = M' \cap \mathcal{L}_P$ and, since $Q \xrightarrow{\mathrm{Sem}} R$, there is $M'' \in \mathrm{Sem}(Q)$ such that $M' = M'' \cap \mathcal{L}_Q$. It follows, since $\mathcal{L}_P \subseteq \mathcal{L}_Q$, that $M = M''\mathcal{L}_P$ for some model $M'' \in \mathrm{Sem}(R)$. Similarly, if $M \in \mathrm{Sem}(R)$ then $M \cap \mathcal{L}_Q \in \mathrm{Sem}(Q)$ and $(M \cap \mathcal{L}_Q) \cap \mathcal{L}_P = M \cap \mathcal{L}_P \in \mathrm{Sem}(P)$. $\square$

Proposition 8. *If $P \xrightarrow{\mathrm{Sem}^*} Q$ and $Q \xrightarrow{\mathrm{Sem}^*} R$ then $P \xrightarrow{\mathrm{Sem}^*} R$.*

Proof. Let T be a program such that $\mathcal{L}_T \cap (\mathcal{L}_R \setminus \mathcal{L}_P) = \emptyset$. Observe that, since $\mathcal{L}_P \subseteq \mathcal{L}_Q \subseteq \mathcal{L}_R$, we have that $\mathcal{L}_T \cap (\mathcal{L}_Q \setminus \mathcal{L}_P) = \mathcal{L}_T \cap (\mathcal{L}_R \setminus \mathcal{L}_Q) = \emptyset$. Then, since $P \xrightarrow{\mathrm{Sem}^*} Q$ and $Q \xrightarrow{\mathrm{Sem}^*} R$, we have that $P \cup T \xrightarrow{\mathrm{Sem}} Q \cup T$ and $Q \cup T \xrightarrow{\mathrm{Sem}} R \cup T$. From Proposition 7 it follows that $P \cup T \xrightarrow{\mathrm{Sem}} R \cup T$. $\square$

The following are other simple propositions, the two particular cases of conservative transformations, that were left out from the paper because of the lack of space.

Lemma 3. *Let P and Q be two theories and let F be a formula such that $\mathcal{L}_{P\cup\{F\}} \cap \mathcal{L}_Q = \emptyset$. If Q is consistent and $P \cup Q \vdash_{\mathrm{X}} F$ then $P \vdash_{\mathrm{X}} F$. Where the logic* X *is either classical or an intermediate logic.*

Proof. Since Q is consistent there is a two-valued model M of Q. In the proof of $P \cup Q \vdash_{\mathrm{X}} F$ replace each atom in $a \in Q$ with $\top$ if $a \in M$ and with $\bot$ if $a \notin M$. Since M is a model of Q all premises in Q will be mapped to tautologies. The restriction $\mathcal{L}_{P\cup\{F\}} \cap \mathcal{L}_Q = \emptyset$ ensures that atoms in P and F will not be altered by this transformation. Thus we end up with a proof for $P \vdash_{\mathrm{X}} F$. $\square$

Proposition 9. *Given a set of atoms A, let $L = \{a \leftarrow a \mid a \in A\}$. If P is a logic program then $P \xrightarrow{\mathrm{AS}} P \cup L$.*

Proof. We will prove that, in particular, the sets $\mathrm{AS}(P) = \mathrm{AS}(P \cup L)$. Let $M \in \mathrm{AS}(P)$ then, by definition, $P \cup \neg(\mathcal{L}_P \setminus M) \cup \neg\neg M \Vdash_{\mathrm{I}} M$. Since the program $P \cup \neg(\mathcal{L}_P \setminus M) \cup \neg\neg M$ is consistent it has a classical model and, if we extend that model evaluating to false all atoms in $L \setminus \mathcal{L}_P$, we obtain that also $P \cup \neg(\mathcal{L}_P \setminus M) \cup \neg(\mathcal{L}_L \setminus \mathcal{L}_P) \cup \neg\neg M$ is consistent. Moreover, since $M \subseteq \mathcal{L}_P$, we have $(\mathcal{L}_P \setminus M) \cup (\mathcal{L}_L \setminus \mathcal{L}_P) = \mathcal{L}_{P\cup L} \setminus M$. So, since L is a set of intuitionistic theorems, we obtain $P \cup L \cup \neg(\mathcal{L}_{P\cup L} \setminus M) \cup \neg\neg M \Vdash_{\mathrm{I}} M$ and, by definition, $M \in \mathrm{AS}(P \cup L)$.

For the converse, if $M \in \mathrm{AS}(P \cup L)$, remove the set of theorems L from the definition of answer sets to obtain $P \cup \neg(\mathcal{L}_{P\cup L} \setminus M) \cup \neg\neg M \Vdash_{\mathrm{I}} M$. Observe now that $M \subseteq \mathcal{L}_P$ since, if we suppose that there is an atom $a \in M$ with $a \notin \mathcal{L}_P$ then a would only appear in the set of premises as $\neg\neg a$ and, by Lemma 3, we end up with $\neg\neg a \vdash_{\mathrm{I}} a$; but this is not possible since $\neg\neg a \rightarrow a$ is not an intuitionistic theorem. It follows again that $(\mathcal{L}_P \setminus M) \cup (\mathcal{L}_L \setminus \mathcal{L}_P) = \mathcal{L}_{P\cup L} \setminus M$ and, therefore, $P \cup \neg(\mathcal{L}_P \setminus M) \cup \neg(\mathcal{L}_L \setminus \mathcal{L}_P) \cup \neg\neg M \Vdash_{\mathrm{I}} M$. Using Lemma 3 again we can remove the set of premises $\neg(\mathcal{L}_L \setminus \mathcal{L}_P)$ and obtain the definition of $M \in \mathrm{AS}(P)$. $\square$

Lemma 4. *Given two disjoint sets of atoms M_1, M_2 and a formula F with $\mathcal{L}_F \subseteq M_1 \cup M_2$, then we have either $M_1 \cup \neg M_2 \vdash_{\mathrm{I}} F$ or $M_1 \cup \neg M_2 \vdash_{\mathrm{I}} \neg F$.*

Sketch of the Proof. A proof can be done by inductively constructing the proof of F. The fact that the premises $M_1 \cup \neg M_2$ can prove either F or $\neg F$ is determined on on the truth value that has the formula F in classical logic when we take the interpretation that makes all atoms in M_1 true and those in M_2 false. The technique required to construct the proof of the formula F in terms of its classical interpretations can be found at (Mendelson 1987). A similar procedure should work here for the intuitionistic logic. $\square$

Lemma 5. *Given two disjoint sets of atoms M_1, M_2 and a formula F with $\mathcal{L}_F \subseteq M_1 \cup M_2$, we have either $\neg\neg M_1 \cup \neg M_2 \vdash_{\mathrm{I}} \neg\neg F$ or $\neg\neg M_1 \cup \neg M_2 \vdash_{\mathrm{I}} \neg F$.*

Proof. From Lemma 4 we know that $M_1 \cup \neg M_2 \vdash_{\mathrm{I}} G$, where G is either F or $\neg F$. Then, for some finite set of atoms $\{a_1, \ldots, a_n\} \subseteq M_1$ we have that $\neg M_2 \vdash_{\mathrm{I}} (a_1 \wedge \cdots \wedge a_n) \rightarrow G$. It follows, using the pair of intuitionistic theorems $\vdash_{\mathrm{I}} A \rightarrow \neg\neg A$ and $\vdash_{\mathrm{I}} \neg\neg(A \rightarrow B) \rightarrow (\neg\neg A \rightarrow \neg\neg B)$, we can then prefix a double negation to G and propagate double negations to obtain $\neg M_2 \vdash_{\mathrm{I}} (\neg\neg a_1 \wedge \cdots \wedge \neg\neg a_n) \rightarrow \neg\neg G$. Again, this is equivalent to the intuitionistic statement $\neg\neg M_1 \cup \neg M_2 \vdash_{\mathrm{I}} \neg\neg G$. □

Proposition 10. *Let $P \in \mathsf{Prp}$. Given a formula F such that $\mathcal{L}_F \subseteq \mathcal{L}_P$ and an atom $x \notin \mathcal{L}_P$, $P \xrightarrow{\mathrm{AS}} P \cup \{x \leftrightarrow F\}$.*

Proof. Take $M \in \mathrm{AS}(P \cup \{x \leftrightarrow F\})$, from the definition of answer sets, we have $P \cup \{x \leftrightarrow F\} \cup \neg(\mathcal{L}_P \setminus M') \cup \neg\neg M' \cup \Delta \Vdash_{\mathrm{I}} M'$, where the set $M' = M \cap \mathcal{L}_P = M \setminus \{x\}$ and $\Delta = \{\neg\neg x\}$ if $x \in M$ or $\Delta = \{\neg x\}$ if $x \notin M$. Also observe, since $\mathcal{L}_F \subseteq \mathcal{L}_P$ and by Lemma 5, that we have $\neg(\mathcal{L}_P \setminus M') \cup \neg\neg M' \vdash_{\mathrm{I}} \neg\neg F$ (or $\neg F$) where, because of consistency, the number of negations should match the number of negations of x in Δ. It follows that $\{x \leftrightarrow F\} \cup \neg(\mathcal{L}_P \setminus M') \cup \neg\neg M' \vdash_{\mathrm{I}} \Delta$ and thus we may remove the set Δ from the premises to obtain $P \cup \{x \leftrightarrow F\} \cup \neg(\mathcal{L}_P \setminus M') \cup \neg\neg M' \Vdash_{\mathrm{I}} M'$. In this statement the atom x now only appears in the premise $x \leftrightarrow F$ and therefore, replacing x with F, we obtain the theorem $F \leftrightarrow F$ that we can also drop from the premises to obtain the definition of $M' \in \mathrm{AS}(P)$.

For the converse take $M \in \mathrm{AS}(P)$ so that, by definition, we have that $P \cup \neg(\mathcal{L}_P \setminus M) \cup \neg\neg M \Vdash_{\mathrm{I}} M$. From Lemma 4 we have $\neg(\mathcal{L}_P \setminus M) \cup M \vdash_{\mathrm{I}} G$, where G is either F or $\neg F$. Let $M' = M \cup \{x\}$ if $G = F$ or $M' = M$ if $G = \neg F$. It also follows, from the original intuitionistic statement, that $P \cup \neg(\mathcal{L}_P \setminus M) \cup \neg\neg M \Vdash_{\mathrm{I}} G$ and therefore, we are able to conclude that $P \cup \{x \leftrightarrow F\} \cup \neg(\mathcal{L}_P \setminus M') \cup \neg\neg M' \Vdash_{\mathrm{I}} M'$, the definition of $M' \in \mathrm{AS}(P)$. □