On Modular Translations and Strong Equivalence

Paolo Ferraris

University of Texas at Austin, Austin TX 78712, USA otto@cs.utexas.edu

Abstract. Given two classes of logic programs, we may be interested in modular translations from one class into the other that are sound wth respect to the answer set semantics. The main theorem of this paper characterizes the existence of such a translation in terms of strong equivalence. The theorem is used to study the expressiveness of several classes of programs, including the comparison of cardinality constraints with monotone cardinality atoms.

1 Introduction

The notion of an answer set (or "stable model"), originally defined in [Gelfond and Lifschitz, 1988], was extended to more general logic programs in various ways. In Fig. 1 we see some examples of extensions of the class of "traditional" rules studied in that paper, and also some subclasses of that class. The language in each line of the table contains the languages shown in the previous lines.

When we compare the expressiveness of two classes of rules R and R', several criteria can be used. First, we can ask whether for any R-program (that is, a set of rules of the type R) one can find a R'-program that has exactly the same answer sets. (That means, in particular, that the R'-program does not use "auxiliary atoms" not occurring in the given R-program.) From this point of view, the classes of rules shown in Fig. 1 can be divided into three groups: a UR- or PR-program has a unique answer set; TR-, TRC- and DR-programs may have many answer sets, but its answer sets always have the "anti-chain" property (one cannot be a proper subset of another); a NDR- or RNE-program can have an arbitrary collection of sets of atoms as its collection of answer sets.

Another comparison criterion is based on the computational complexity of the problem of the existence of an answer set. We pass, in the complexity hierarchy, from P in case of UR- and PR-programs, to NP in case of TR- and TRC-programs [Marek and Truszczyński, 1991], and finally to Σ_2^P for more complex kinds of programs [Eiter and Gottlob, 1993].

A third criterion consists in checking whether every rule in R is strongly equivalent [Lifschitz *et al.*, 2001] to a R'-program. From this point of view, PR is essentially more expressive than UR: we will see at the end of Sect. 3 that $a \leftarrow b, c$ is not strongly equivalent to any set of unary rules. Furthermore, TRC and DR are essentially different from each other, since no program in TRC is strongly equivalent to the rule $p; q$ in DR [Turner, 2003, Proposition 1].

class of rules	syntactic form
UR	unary rules: $a \leftarrow$ (also written as simply a) and $a \leftarrow b$
PR	positive rules: $a \leftarrow b_1, \ldots, b_n$
TR	traditional rules: $a \leftarrow b_1, \ldots, b_n, not\ c_1, \ldots, not\ c_m$
TRC	TRs + constraints: TRs and $\leftarrow b_1, \ldots, b_n, not\ c_1, \ldots, not\ c_m$
DR	disjunctive rules: $a_1; \ldots; a_p \leftarrow b_1, \ldots, b_n, not\ c_1, \ldots, not\ c_m$
NDR	negational disjunctive rules: $a_1, \ldots; a_p; not\ d_1; \ldots; not\ d_q \leftarrow b_1, \ldots, b_n, not\ c_1, \ldots, not\ c_m$
RNE	rules with nested expressions: $F \leftarrow G$

Fig. 1. A classification of logic programs under the answer set semantics. Here a, b, c, d stand for propositional atoms. F, G stand for nested expressions without classical negation [Lifschitz *et al.*, 1999], that is, expressions formed from atoms, $\top$ and $\bot$, using conjunction (,), disjunction (;) and negation as failure (not).

A fourth comparison criterion is based on the existence of a translation from R-programs to R'-programs that is not only sound (that is, preserves the program's answer sets) but is also modular: it can be applied to a program rule-by-rule. For instance, [Janhunen, 2000] showed that there is no modular translation from PR to UR and from TR to PR [1]. On the other hand, RNE can be translated into NDR by a modular procedure similar to converting formulas to conjunctive normal form [Lifschitz *et al.*, 1999].

The main theorem of this paper shows that under some general conditions, the last two criteria — the one based on strong equivalence and the existence of a sound modular translation — are equivalent to each other. This offers a method to prove that there is no modular translation from R to R' by finding a rule in R that is not strongly equivalent to any R'-program. For instance, in view of the Proposition 1 from [Turner, 2003] mentioned above, no modular translation exists from DR to TRC.

To apply the main theorem to other cases, we need to learn more about the strong equivalence relations between a single rule of a language and a set of rules. We show that for many rules r in NDR, any NDR-program that is strongly equivalent to r contains a rule that is at least as "complex" as r. This fact will allow us to conclude that all classes UR, PR, TR, TRC, DR and NDR are essentially different from each other in terms of strong equivalence. In view of the main theorem, it follows that they are essentially different from each other in the sense of the modular translation criterion as well.

[1] His results are actually stronger, see Sect. 7 below.

Finally, we show how to apply our main theorem to programs with weight constraints [Niemelä and Simons, 2000]. As a result, we find that it is not possible to translate programs with weight constraints into programs with monotone cardinality atoms [Marek and Niemelä, 2004] in a modular way (unless the translation introduces auxiliary atoms).

The paper continues with the statement of our main theorem (Sect. 2). In Sect. 3, we study the expressiveness of subclasses of NDR in terms of strong equivalence and modular translations. We move to the study of cardinality constraints in Sect. 4. Section 5 provides some background needed for the proof of some of the claims of this paper (Sect. 6).

2 Modular transformations and strong equivalence

We assume that the reader is familiar with the concept of an answer set for the classes of logic programs in Fig. 1 (the semantics for the class RNE, which is applicable to all its subclasses, is reproduced in Sect. 5.1). A *program* is a subset of RNE. Two programs Π_1 and Π_2 are *strongly equivalent* if, for every program Π, $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same answer sets. A *(modular) transformation* is a function f such that

- $\mathrm{Dom}(f) \subseteq \mathrm{RNE}$, and
- for every rule $r \in \mathrm{Dom}(f)$, $f(r)$ is a program such that every atom occurring in it occurs in r also.

A transformation f is *sound* if, for every program $\Pi \subseteq \mathrm{Dom}(f)$, Π and $\bigcup_{r\in\Pi} f(r)$ have the same answer sets.

For example, the transformation defined in the proof of Proposition 7 from [Lifschitz *et al.*, 1999], which eliminates nesting from a program with nested expressions, is a sound transformation. For instance, for this transformation f,

$$f(a \leftarrow b; c) = \{a \leftarrow b,\ a \leftarrow c\}.$$

As another example of a sound transformation, consider the transformation f with $\mathrm{Dom}(f) = NDR$, where

$$f(not\ d_1; \ldots; not\ d_q \leftarrow b_1, \ldots, b_n, not\ c_1, \ldots, not\ d_m) = \\ \{\leftarrow b_1, \ldots, b_n, d_1, \ldots, d_q, not\ c_1, \ldots, not\ d_m\}$$

and $f(r) = \{r\}$ for the other rules r in NDR. On the other hand, the familiar method of eliminating constraints from a program that turns $\leftarrow p$ into $q \leftarrow p, not\ q$ is not a transformation in the sense of our definition, because it introduces an atom q that doesn't occur in $\leftarrow p$.

This is the theorem that relates strong equivalence and modular transfomations:

Theorem 1 (Main Theorem). *For every transformation f such that* $\mathrm{Dom}(f)$ *contains all unary rules, f is sound iff, for each $r \in R$, $f(r)$ is strongly equivalent to r.*

Our definition of transformation requires that all atoms that occur in $f(r)$ occur in r also. The following counterexample shows that without this assumption the assertion of the Main Theorem would be incorrect. Let p and q be two atoms, and, for each rule $r = F \leftarrow G$ in RNE, let $f_1(r)$ be

$$\begin{array}{l} F \leftarrow G, not\ p \\ F \leftarrow G, not\ q \\ F_{p \leftrightarrow q} \leftarrow G_{p \leftrightarrow q}, not\ not\ p, not\ not\ q. \end{array}$$

where $F_{p \leftrightarrow q}$ and $G_{p \leftrightarrow q}$ stand for F and G with all the occurrences of p replaced by q and vice versa. Note that f_1 is not a transformation as defined in this paper since q occurs in $f_1(p \leftarrow \top)$. It can also be shown that $p \leftarrow \top$ and $f_1(p \leftarrow \top)$ are not strongly equivalent. However, the "transformation" is sound:

Proposition 1. *For any program Π, Π and $\bigcup_{r \in \Pi} f_1(r)$ have the same answer sets.*

Without the assumption that $\mathrm{UR} \subseteq \mathrm{Dom}(f)$, the Main Theorem would not be correct either. We define a transformation f_2 such that $\mathrm{Dom}(f_2)$ consists of all rules of DR where all atoms in the body are prefixed by negation as failure, and the head is nonempty: the rules have the form

$$a_1; \ldots; a_p \leftarrow not\ c_1, \ldots, not\ c_m. \tag{1}$$

with $p > 0$. For each rule r of the form (1), $f_2(r)$ is defined as

$$\{a_i \leftarrow not\ a_1, \ldots, not\ a_{i-1}, not\ a_{i+1}, \ldots, not\ a_p, not\ c_1, \ldots, not\ c_m\ :\ 1 \leq i \leq p\}.$$

It is easy to see that $f_2(p; q)$ is not strongly equivalent to $p; q$. However, this transformation is sound:

Proposition 2. *For any program $\Pi \subseteq \mathrm{Dom}(f_2)$, Π and $\bigcup_{r \in \Pi} f_2(r)$ have the same answer sets.*

3 Applications: Negational Disjunctive Rules

In order to apply the Main Theorem to modular translations, we first need to study some properties of strong equivalence. We focus on the class NDR.

If r is

$$a_1, \ldots; a_p; not\ d_1; \ldots; not\ d_q \leftarrow b_1, \ldots, b_n, not\ c_1, \ldots, not\ c_m$$

define

$$\begin{array}{ll} head^+(r) = \{a_1, \ldots, a_p\} & head^-(r) = \{d_1, \ldots, d_q\} \\ body^+(r) = \{b_1, \ldots, b_n\} & body^-(r) = \{c_1, \ldots, c_m\}. \end{array}$$

We say that r is *basic* if every pair of these sets, except possibly for the pair $head^+(r)$, $head^-(r)$, is disjoint.

Any nonbasic rule can be easily simplified: it is either strongly equivalent to the empty program or contains redundant terms in the head. A basic rule, on the other hand, cannot be simplified if it contains at least one nonnegated atom in the head:

Theorem 2. *Let r be a basic rule in NDR such that $head^+(r) \neq \emptyset$. Every program subset of NDR that is strongly equivalent to r contains a rule r' such that*

$$head^+(r) \subseteq head^+(r') \qquad body^+(r) \subseteq body^+(r')$$
$$head^-(r) \subseteq head^-(r') \qquad body^-(r) \subseteq body^-(r')$$

This theorem shows us that for most basic rules r (the ones with at least one positive element in the head), every program strongly equivalent to r must contain a rule that is at least as "complex" as r.

Given two subsets R and R' of RNE, a *(modular) translation* from R to R' is a transformation f such that $\mathrm{Dom}(f) = R$ and $f(r)$ is a subset of R' for each $r \in \mathrm{Dom}(f)$. Using Theorems 1 and 2, we can differentiate between the classes of rules in Fig. 1 in terms of modular translations:

Proposition 3. *For any two languages R and R' among UR, PR, TRC, TR, DR and NDR such that $R' \subset R$, there is no sound translation from R to R'.*

Theorems 1 and 2 allow us also to differentiate between subclasses of NDR described in terms of the sizes of various parts of the rule. Define, for instance PBR_i ("positive body of size i") as the set of rules r of NDR such that $|body^+(r)| \leq i$. We can show that, for every $i \geq 0$, there is no sound translation from PBR_{i+1} to PBR_i (or even from $\mathrm{PBR}_{i+1} \cap \mathrm{PR}$ to PBR_i). Similar properties can be stated in terms of the sizes of $body^-(r)$, $head^+(r)$ and $head^-(r)$.

Another consequence of Theorem 2 is in terms of (absolute) tightness of a program [Erdem and Lifschitz, 2003, Lee, 2005]. Tightness is an important property of logic programs: if a program is tight then its answer sets can be equivalently characterized by the satisfaction of a set of propositional formulas of about the same size. Modular translations usually don't make nontight programs tight:

Proposition 4. *Let R be any subset of NDR that contains all unary rules, and let f be any sound translation from R to NDR. For every nontight program $\Pi \subset R$ consisting of basic rules only, $\bigcup_{r \in \Pi} f(r)$ is nontight.*

4 Applications: programs with cardinality constraints

4.1 Syntax

We briefly review the syntax of programs with cardinality constraints [Niemelä and Simons, 2000].

A *rule element* is an atom possibly prefixed with negation as failure symbol *not*. A *cardinality constraint* is an expression of the form

$$L\{c_1, \ldots, c_m\}U \tag{2}$$

where

– each of L, U is (a symbol for) an integer or $-\infty$, $+\infty$,
– $m \geq 0$, and
– $c_1, \ldots, c_m$ are rule elements.

As an abbreviation, L can be omitted if $L = -\infty$; similarly, we can drop U if $U = +\infty$. A *rule with cardinality constraints* is an expression of the form

$$C_0 \leftarrow C_1, \ldots, C_n \tag{3}$$

where $C_0, \ldots, C_n$ ($n \geq 0$) are cardinality constraints. A *program with cardinality constraints* is a set of rules with cardinality constraints.

Let CCR denote the set of all rules with cardinality constraints. A straightforward generalization of the definition of a transformation allows us to talk about sound translations between subclasses of CCR, and also between a class of CCR and a subclass of RNE. The concept of (modular) transformations and translations can be extended to programs with cardinality constraints: we can have translations between subclasses of CCR, and from/to subclasses of RNE. The definition of the soundness for those translations follows as well.

Another class of programs similar to the one with cardinality constraints — programs with monotone cardinality atoms — has been defined in [Marek and Niemelä, 2004]. The results of that paper show that rules with monotone cardinality atoms are essentially identical to rules with cardinality constraints that don't contain negation as failure; we will denote the set of all such rules by PCCR ("positive cardinality constraints").

4.2 Translations

First of all, we show how programs with cardinality constraints are related to the class NDR. Let SNDR (Simple NDR) be the language consisting of rules of the form

$$a; not\ d_1; \ldots; not\ d_q \leftarrow b_1, \ldots, b_n, not\ c_1, \ldots, not\ c_m \tag{4}$$

and

$$\leftarrow b_1, \ldots, b_n, not\ c_1, \ldots, not\ c_m. \tag{5}$$

Proposition 5. *There exist sound translations from SNDR to CCR, and back.*

If we don't allow negation in cardinality constraints, another relationship holds. We define the class VSNDR (Very Simple NDR) consisting of rules of the form

$$a; not\ a \leftarrow b_1, \ldots, b_n, not\ c_1, \ldots, not\ c_m \tag{6}$$

and of the form (5).

Proposition 6. *There exist sound translations from VSNDR to PCCR, and back.*

Using Theorems 1 and 2, we can prove:

Proposition 7. *There is no sound translation from CCR to PCCR.*

Since the class PCCR is essentially identical to the class of rules with monotone cardinality atoms, we have that programs with cardinality constraints are essentially more expressive than programs with monotone cardinality atoms.

5 Background for proofs

5.1 Answer set semantics for RNE

The semantics of programs is characterized by defining when a set X of atoms is an answer set for a program Π. As a preliminary step, we define when a set X of atoms *satisfies* a formula F (symbolically, $X \models F$), as follows:

- for an atom a, $X \models a$ if $a \in X$
- $X \models \top$
- $X \not\models \bot$
- $X \models (F, G)$ if $X \models F$ and $X \models G$
- $X \models (F; G)$ if $X \models F$ or $X \models G$
- $X \models \mathit{not}\ F$ if $X \not\models F$.

We say that X *satisfies* a program Π (symbolically, $X \models \Pi$) if, for every rule $F \leftarrow G$ in Π, $X \models F$ whenever $X \models G$.

The *reduct* Π^X of a program Π with respect to a set X of atoms is obtained by replacing each outermost formula of the form $\mathit{not}\ F$ (that is, every formula of the form $\mathit{not}\ F$ not in the scope of negation as failure) by $\bot$, if $X \models F$, and by $\bot$ otherwise.

The concept of an answer set is defined first for programs not containing negation as failure: a set X of atoms is an *answer set* for such a program Π if X is a minimal set satisfying Π. For an arbitrary program Π, we say that X is an *answer set* for Π if X is an answer set for the reduct Π^X.

5.2 Strong equivalence

The following lemma is the main criterion that we use to check strong equivalence in most of the proofs. It can be proved in a way similar to the equivalence criterion from [Turner, 2003].

Lemma 1. *Let A be the set of atoms occurring in programs Π_1 and Π_2. Π_1 and Π_2 are strongly equivalent iff, for each $Y \subseteq A$,*

- *$Y \models \Pi_1^Y$ iff $Y \models \Pi_2^Y$, and*
- *if $Y \models \Pi_1^Y$ then, for each $X \subset Y$, $X \models \Pi_1^Y$ iff $X \models \Pi_2^Y$.*

Next lemma can be easily proven under the characterization of strong equivalence as stated in [Lifschitz *et al.*, 2001].

Lemma 2. *Let P_1 and P_2 be sets of programs. If each program in P_1 is strongly equivalent to a program in P_2 and vice versa, then $\bigcup_{\Pi \in P_1} \Pi$ and $\bigcup_{\Pi \in P_2} \Pi$ are strongly equivalent.*

Finally, we will use another property of strong equivalence from [Lifschitz *et al.*, 2001]:

Lemma 3. *Two programs P_1 and P_2 are strongly equivalent iff, for every program $\Pi \subseteq UR$, $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same answer sets.*

6 Proofs

6.1 Proof of the Main Theorem

Main Theorem. *For every transformation f such that $\mathrm{Dom}(f)$ contains all unary rules, f is sound iff, for each $r \in R$, $f(r)$ is strongly equivalent to r.*

The proof from right to left is a direct consequence of Lemma 2: if $f(r)$ is strongly equivalent to r for every rule $r \in R$, then for any $\Pi \subseteq R$, Π and $\bigcup_{r \in \Pi} f(r)$ are strongly equivalent, and consequently have the same answer sets.

In the proof from left to right, we first consider the case when r is a unary rule, and then extend the conclusion to arbitrary rules. In the rest of this section, f is an arbitrary sound transformation such that $\mathrm{Dom}(f)$ contains all unary rules. By a and b we denote distinct atoms.

Lemma 4. *For every fact a, $\{a\}$ and $f(a)$ are strongly equivalent.*

Proof. For every program Π that has $\{a\}$ as the only answer set, we have that $\emptyset \not\models \Pi^{\emptyset}$, $\{a\} \models \Pi^{\{a\}}$ and $\emptyset \not\models \Pi^{\{a\}}$. Since $\{a\}$ and $f(r)$ are two of such programs Π, and a is the only atom that occurs in r and $f(r)$, we can conclude that $\{r\}$ and $f(r)$ are strongly equivalent by Lemma 1. □

Lemma 5. *For every rule r and fact a, $\{r, a\}$ and $f(r) \cup \{a\}$ have the same answer sets.*

Proof. In view of Lemma 4, $f(r) \cup \{a\}$ and $f(r) \cup f(a)$ have the same answer sets, and the same holds for $\{r, a\}$ and $f(r) \cup f(a)$ by hypothesis. □

Lemma 6. *For every rule r of the form $a \leftarrow a$,*

(i) $\emptyset \models f(r)^{\emptyset}$,
(ii) $\{a\} \models f(r)^{\{a\}}$, *and*
(iii) $\emptyset \models f(r)^{\{a\}}$.

Proof. First of all, since the empty set in the only answer set for $\{r\}$ and then for $f(r)$, (i) is clearly true. Now consider the program consisting of rule r plus fact a. Since $\{a\}$ is an answer set for $\{r, a\}$, it is an answer set for $f(r) \cup \{a\}$ also by Lemma 5. Consequently, $\{a\} \models (f(r) \cup \{a\})^{\{a\}}$, which proves (ii). From (ii) and the fact that $\{a\}$ is not an answer set for $f(r)$, (iii) follows also. □

Lemma 7. *For every rule r of the form $a \leftarrow b$,*

(i) $\emptyset \models f(r)^{\emptyset}$,
(ii) $\{a\} \models f(r)^{\{a\}}$,
(iii) $\emptyset \models f(r)^{\{a\}}$, *and*
(iv) $\{b\} \not\models f(r)^{\{b\}}$.

Proof. The proof of the first three claims is similar to the one of Lemma 6. To prove (iv), consider that since $\{b\}$ is not an answer set for $\{r, b\}$, it is not an answer set for $f(r) \cup \{b\}$ either by Lemma 5. But $\emptyset \not\models (f(r) \cup \{b\})^{\{b\}}$ because $\emptyset \not\models \{b\}^{\{b\}}$; consequently $\{b\} \not\models (f(r) \cup \{b\})^{\{b\}}$. Since $\{b\} \models \{b\}^{\{b\}}$, we can conclude (iv). □

Lemma 8. *For every rule r of the form $a \leftarrow b$,*

(i) $\{a, b\} \models f(r)^{\{a,b\}}$,
(ii) $\{b\} \not\models f(r)^{\{a,b\}}$, *and*
(iii) $\{a\} \models f(r)^{\{a,b\}}$.

Proof. Set $\{a, b\}$ is an answer set for $\{r, b\}$, and consequently for $f(r) \cup \{b\}$ also by Lemma 5. Consequently $\{a, b\} \models (f(r) \cup \{b\})^{\{a,b\}}$ — from which we derive (i) — and all proper subsets of $\{a, b\}$ don't satisfy $(f(r) \cup \{b\})^{\{a,b\}}$. Since $\{b\} \models \{b\}^{\{a,b\}}$, we have that (ii) holds. Notice that $\{a, b\} \models (f(r) \cup \{a\})^{\{a,b\}}$ follows from (i), and that $\{a, b\}$ is not an answer set for $f(r) \cup \{a\}$ because it is not an answer set for $\{r, a\}$ and by Lemma 5. Consequently there is a proper subset of $\{a, b\}$ that satisfies $(f(r) \cup \{a\})^{\{a,b\}}$. Such subset can only be $\{a\}$ because it is the only one that satisfies $\{a\}^{\{a,b\}}$. We can conclude (iii). □

Lemma 9.

$$\emptyset \models f(a \leftarrow b)^{\{a,b\}}.$$

Proof. Let r be $a \leftarrow b$, and r' be $b \leftarrow a$. Lemma 8 can help us determine which subsets of $\{a, b\}$ satisfy $(f(r) \cup f(r'))^{\{a,b\}}$: from part (i) applied to both r and r', we get that $\{a, b\}$ satisfies this program, while $\{b\}$ (by part (ii) applied to r) and $\{a\}$ (by part (ii) applied to r') don't. On the other hand $\{a, b\}$ is not an answer set for $\{r, r'\}$ and then for $f(r) \cup f(r')$ by the soundness hypothesis. We can conclude that $\emptyset \models (f(r) \cup f(r'))^{\{a,b\}}$ from which the lemma's assertion follows. □

Lemma 10. *For every unary program Π, Π and $\bigcup_{r \in \Pi} f(r)$ are strongly equivalent.*

Proof. In view of Lemma 2, it is sufficient to show that for each unary rule r, $\{r\}$ and $f(r)$ are strongly equivalent. For rules that are facts, this is shown by Lemma 4. For rules r of the form $a \leftarrow a$, in view of Lemma 6 it is easy to check that, for every sets X and Y such that $X \subseteq Y \subseteq \{a\}$, $X \models (a \leftarrow a)^Y$ iff $X \models f(a \leftarrow a)^Y$. So $a \leftarrow a$ and $f(a \leftarrow a)$ are strongly equivalent by Lemma 1. Similarly, we can check that for every sets X and Y such that $X \subseteq Y \subseteq \{a, b\}$ with $X \neq \emptyset$ or $Y \neq \{b\}$, that $X \models (a \leftarrow b)^Y$ iff $X \models f(a \leftarrow b)^Y$. This is by Lemmas 7–9. Consequently $a \leftarrow b$ and $f(a \leftarrow b)$ are strongly equivalent by Lemma 1 as well. □

Now we are ready to prove the second part of the main theorem: for any rule $r \in \mathrm{Dom}(f)$, $f(r)$ and $\{r\}$ are strongly equivalent. By Lemma 3,it is sufficient to show that, for each unary program Π, $\Pi \cup \{r\}$ and $\Pi \cup f(r)$ have the same answer sets. First we notice that $\Pi \cup \{r\}$ and $\bigcup_{r' \in \Pi \cup \{r\}} f(r')$ have the same answer sets since $\Pi \cup \{r\} \subseteq \mathrm{Dom}(f)$ and for the soundness of the transformation. Then we can see that $\bigcup_{r' \in \Pi \cup \{r\}} f(r') = \bigcup_{r' \in \Pi} f(r') \cup f(r)$. Finally, $\bigcup_{r' \in \Pi} f(r') \cup f(r)$ and $\Pi \cup f(r)$ have the same answer sets because, by Lemma 10, programs Π and $\bigcup_{r' \in \Pi} f(r')$ are strongly equivalent. □

6.2 Proof of Propositions 1 and 2

Proposition 1. *For any program Π, Π and $\bigcup_{r \in \Pi} f_1(r)$ have the same answer sets.*

Proof. (outline) Consider any set of atoms X. If $\{p, q\} \not\subseteq X$ then $f(F \leftarrow G)^X$ is essentially $\{F \leftarrow G\}^X$, and the claim easily follows. Otherwise $f_1(F \leftarrow G)^X$ is essentially $\{F_{p \leftrightarrow q} \leftarrow G_{p \leftrightarrow q}\}$. If we extend the notation of the subscript $p \leftrightarrow q$ to both programs and sets of atoms, $(\bigcup_{r \in \Pi} f_1(r))^X$ can be seen as $(\Pi^X)_{p \leftrightarrow q}$. A subset Y of X satisfies Π^X iff $Y_{p \leftrightarrow q}$ satisfies $(\Pi^X)_{p \leftrightarrow q}$. Since $Y_{p \leftrightarrow q} \subseteq X$ and $|Y_{p \leftrightarrow q}| = |Y|$, we can conclude that X is a minimal set satisfying $(\bigcup_{r \in \Pi} f_1(r))^X$ iff it is a minimal set satisfying Π^X. □

Proposition 2. *For any program $\Pi \subseteq$ NBR, Π and $\bigcup_{r \in \Pi} f_2(r)$ have the same answer sets.*

Proof. (outline) Let Π' be $\bigcup_{r \in \Pi} f_2(r)$. The proof consists in seeing that that Π and Π' are both absolutely tight. So each set X atoms is an answer set for Π iff X satisfies Π and X is "supported" by Π, and similarly for Π' (for more details, see [Lee, 2005]). Those conditions are satisfied, for both Π and Π', by the same sets of atoms.

It is not hard to see that the same sets X satisfy both conditions for Π and Π'. □

6.3 Proof of Theorem 2

For simplicity, we consider the definition of an answer set for NDR programs as defined in [Lifschitz and Woo, 1992], in which the reduct Π^X consists of the rule

$$head^+(r) \leftarrow body^+(r) \tag{7}$$

($head^+(r)$ here stands for the disjunction of its elements, $body^+(r)$ for their conjunction) for every rule r in Π such that $head^-(r) \subseteq X$ and $body^-(r) \cap X = \emptyset$. Since Π^X is satisfied by the same sets of atoms regardless on the definition, the strong equivalent criterion based on satisfaction of the reduct doesn't change.

Let Π be a program strongly equivalent to r. Let X be $head^+(r) \cup head^-(r) \cup body^+(r)$, and let Y be $body^+(r)$. Then r^X is (7). By Lemma 1, since $X \models r^X$ (recall that $head^+(r)$ is nonempty by hypothesis) then $X \models \Pi^X$, and since

$Y \not\models r^X$ it follows that $Y \not\models \Pi^X$. Consequently, there is a rule of Π — the rule r' of the theorem's statement — such that $X \models (r')^X$ and $Y \not\models (r')^X$. From this second fact, $(r')^X$ is nonempty so it is

$$head^+(r') \leftarrow body^+(r'), \tag{8}$$

and also $Y \models body^+(r')$ and $Y \not\models head^+(r')$.

To prove that $head^+(r) \subseteq head^+(r')$, take any atom $a \in head^+(r)$. The set $Y \cup \{a\}$ satisfies r^X, so it satisfies Π^X by Lemma 1, and then $(r')^X$ also. On the other hand, since $Y \models body^+(r')$ and $Y \subseteq Y \cup \{a\}$, we have that $Y \cup \{a\} \models body^+(r')$. Consequently, $Y \cup \{a\} \models head^+(r')$. Since $Y \not\models head^+(r')$, we can conclude that a is an element of $head^+(r')$.

The proof that $body^+(r) \subseteq body^+(r')$ is similar to the previous part of the proof, by taking any $a \in body^+(r)$ and considering the set $Y \setminus \{a\}$ instead of $Y \cup \{a\}$.

To prove that $head^-(r) \subseteq head^-(r')$, take any atom $a \in head^-(r)$. Since $r^{X\setminus\{a\}}$ is empty, it is satisfied, in particular, by Y and $X \setminus \{a\}$. Consequently, $Y \models (r')^{X\setminus\{a\}}$ by Lemma 1. On the other hand, $Y \not\models (r')^X$, so $(r')^{X\setminus\{a\}}$ is not (8), and then it is empty. The only case in which $(r')^{X\setminus\{a\}}$ is empty and $(r')^X$ is not is if $a \in head^-(r')$.

The proof that $body^-(r) \subseteq body^-(r')$ is similar to the previous part of the proof, by taking any $a \in body^-(r)$ and considering the reduct $r^{X\cup\{a\}}$.

6.4 Proofs of Propositions 5–7 (outline)

In the proof of Proposition 5, from SNDR to CCR, we take the following sound translation f: if r has the form (4) then $f(r)$ is

$$1\{a\} \leftarrow 1\{b_1\}, \ldots, 1\{b_n\}, \{c_1\}0, \ldots, \{c_m\}0, \{not\ d_1\}0, \ldots, \{not\ d_q\}0,$$

and, if r has the form (5), then $f(r)$ is

$$1\{\} \leftarrow 1\{b_1\}, \ldots, 1\{b_n\}, \{c_1\}0, \ldots, \{c_m\}0, 1\{d_1\}, \ldots, 1\{d_q\}.$$

A sound translation f from VSNDR to PCCR (proof of Proposition 6) is defined as follows: if r has the form (6) then $f(r)$ is

$$\{a\} \leftarrow 1\{b_1\}, \ldots, 1\{b_n\}, \{c_1\}0, \ldots, \{c_m\}0$$

while for rules r of the form (5), $f(r)$ is the same as in the previous translation.

The proof in the other direction is based on the modular translation from programs with cardinality constraints to programs with nested expressions whose heads are atoms or $\bot$, as defined in [Ferraris and Lifschitz, 2005]. If we first apply such translation to any CCR-program, then the one from RNE to NDR of [Lifschitz *et al.*, 1999], we get a SNDR-program. Similarly from a PCCR-program we get a VSNDR-program.

Proposition 7 follows from Propositions 5 and 6, the fact that no subset of VSNDR is strongly equivalent to the rule a; *not* a; *not* b of SNDR by Theorem 2, and the Main Theorem.

7 Conclusions

We have established a relationship between modular transformations and strong equivalence. We showed how it can be used to determine whether sound modular translations between languages are possible.

Other definitions of a modular translation allow the the introduction of auxiliary atoms. This is, for instance, the case for the definitions in [Ferraris, 2005] and [Janhunen, 2000]. These two papers are also different from the work described in this note in that they take into account the computation time of translation algorithms.

We restricted, in Sect. 2, the domain and range of transformations to programs with nested expressions. If we drop this limitation by allowing arbitrary propositional formulas, and we define the soundness of a transformation in terms of equilibrium logic [Pearce, 1997, 1999] then the Main Theorem will still hold. Since each propositional theory is strongly equivalent to a logic program [Cabalar and Ferraris, 2004] we can conclude that there exists a sound and modular translation from propositional theories to RNE and vice versa.

Criteria for strong equivalence, in part related to Theorem 2, are proposed in [Lin and Chen, 2005].

The theorems about cardinality constraints stated in Sect. 4 can be trivially extended to arbitrary weight constraints in view of the fact that an expression $c = w$ in a weight constraint can always be replaced by w copies of $c = 1$.

Acknowledgments

I am grateful to Joohyung Lee for comments on this work. Special thanks go to Vladimir Lifschitz for many comments and discussions on the topic, and his careful reading of this paper. This research was partially supported by the National Science Foundation under Grant IIS-0412907.

References

[Cabalar and Ferraris, 2004] Pedro Cabalar and Paolo Ferraris. Propositional theories are equivalent to logic programs. In preparation, 2004.

[Eiter and Gottlob, 1993] Thomas Eiter and Georg Gottlob. Complexity results for disjunctive logic programming and application to nonmonotonic logics. In Dale Miller, editor, *Proceedings of International Logic Programming Symposium (ILPS)*, pages 266–278, 1993.

[Erdem and Lifschitz, 2003] Esra Erdem and Vladimir Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, 3:499–518, 2003.

[Ferraris and Lifschitz, 2005] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5:45–74, 2005.

[Ferraris, 2005] Paolo Ferraris. A modular, polynomial method for eliminating weight constraints. [2] Submitted to the same conference, 2005.

[2] `http://www.cs.utexas.edu/users/otto/papers/newweight.ps` .

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080, 1988.

[Janhunen, 2000] Tomi Janhunen. Comparing the expressive powers of some syntactically restricted classes of logic programs. In *Proc. 1st International Conference on Computational Logic*, volume 1861, pages 852–866, 2000.

[Lee, 2005] Joohyung Lee. A model-theoretic counterpart of loop formulas. In *Proc. IJCAI*, 2005. To appear.

[Lifschitz and Woo, 1992] Vladimir Lifschitz and Thomas Woo. Answer sets in general nonmonotonic reasoning (preliminary report). In Bernhard Nebel, Charles Rich, and William Swartout, editors, *Proc. Third Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 603–614, 1992.

[Lifschitz *et al.*, 1999] Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.

[Lifschitz *et al.*, 2001] Vladimir Lifschitz, David Pearce, and Agustin Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:526–541, 2001.

[Lin and Chen, 2005] Fangzhen Lin and Yin Chen. Discovering classes of strongly equivalent logic programs. In *Proc. IJCAI*, 2005. To appear.

[Marek and Niemelä, 2004] Victor Marek and Ilkka Niemelä. On logic programs with cardinality constraints. In *Proc. 7th Int'l Conference on Logic Programming and Nonmonotonic Reasoning*, pages 154–166, 2004.

[Marek and Truszczyński, 1991] W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38(3):588–619, 1991.

[Niemelä and Simons, 2000] Ilkka Niemelä and Patrik Simons. Extending the Smodels system with cardinality and weight constraints. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 491–521. Kluwer, 2000.

[Pearce, 1997] David Pearce. A new logical characterization of stable models and answer sets. In Jürgen Dix, Luis Pereira, and Teodor Przymusinski, editors, *Non-Monotonic Extensions of Logic Programming (Lecture Notes in Artificial Intelligence 1216)*, pages 57–70. Springer-Verlag, 1997.

[Pearce, 1999] David Pearce. From here to there: Stable negation in logic programming. In D. Gabbay and H. Wansing, editors, *What Is Negation?* Kluwer, 1999.

[Turner, 2003] Hudson Turner. Strong equivalence made easy: nested expressions and weight constraints. *Theory and Practice of Logic Programming*, 3(4,5):609–622, 2003.