Weight Constraints as Nested Expressions

PAOLO FERRARIS and VLADIMIR LIFSCHITZ

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA
(*e-mail:* {otto,vl}@cs.utexas.edu)

Abstract

We compare two recent extensions of the answer set (stable model) semantics of logic programs. One of them, due to Lifschitz, Tang and Turner, allows the bodies and heads of rules to contain nested expressions. The other, due to Niemelä and Simons, uses weight constraints. We show that there is a simple, modular translation from the language of weight constraints into the language of nested expressions that preserves the program's answer sets. Nested expressions can be eliminated from the result of this translation in favor of additional atoms. The translation makes it possible to compute answer sets for some programs with weight constraints using satisfiability solvers, and to prove the strong equivalence of programs with weight constraints using the logic of here-and-there.

KEYWORDS: answer sets, cardinality constraints, SMODELS, stable models, weight constraints.

1 Introduction

The notion of an answer set (or "stable model") was defined in (Gelfond and Lifschitz 1988) for logic programs whose rules have simple syntactic structure. The head of such a rule is an atom. The body is a list of atoms, some of them possibly prefixed with the negation as failure symbol. In this paper, we compare two recent extensions of the answer set semantics.

In one of these extensions, the head and the body of a rule are allowed to contain negation as failure (*not*), conjunction (,) and disjunction (;), nested arbitrarily (Lifschitz *et al.* 1999). In particular, negation as failure can occur in the head of a rule, as proposed in (Lifschitz and Woo 1992). For instance,

$$a; not\ a \tag{1}$$

is a rule with the empty body. The program whose only rule is (1) can be shown to have two answer sets: $\emptyset$ and $\{a\}$. The rule

$$a \leftarrow not\ not\ a \tag{2}$$

is another example of a rule with nested expressions. It is "nondisjunctive"—its head is an atom; but its body contains nested occurrences of negation as failure. The program whose only rule is (2) has the same answer sets as (1).

According to the second proposal (Niemelä and Simons 2000), rules are allowed to contain "cardinality constraints" and more general "weight constraints." For instance,

$$0 \leq \{a, b\} \leq 1 \tag{3}$$

is a cardinality constraint. This expression can be viewed as a logic program consisting of a single rule with the empty body. Its answer sets are the subsets of $\{a, b\}$ whose cardinalities are between 0 and 1, that is to say, sets $\emptyset$, $\{a\}$ and $\{b\}$.

Cardinality and weight constraints are important elements of the input language of SMODELS — a software system for computing answer sets that can be used to solve many kinds of combinatorial search problems.[1] The idea of this programming method, called *answer set programming*, is to represent the given search problem by a logic program whose answer sets correspond to solutions. Cardinality constraints are found in many programs of this kind.

It may appear that the two extensions of the basic syntax of logic programs — nested expressions and weight constraints — have little in common. The following observation suggests that it would not be surprising actually if these ideas were related to each other. The original definition of an answer set is known to have the "anti-chain" property: an answer set for a program cannot be a subset of another answer set for the same program. Examples (1) and (2) show that the anti-chain property is lost as soon as nested expressions are allowed in rules. Example (3) shows that in the presence of cardinality constraints the anti-chain property does not hold either.

In this paper we show that there is indeed a close relationship between these two forms of the answer set semantics: cardinality and weight constraints can be viewed as shorthand for nested expressions of a special form. We define a simple, modular translation that turns any program Ω with weight constraints into a program $[\Omega]$ with nested expressions that has the same answer sets as Ω. Furthermore, every rule of $[\Omega]$ can be equivalently replaced with a set of nondisjunctive rules, and this will lead us to a nondisjunctive version $[\Omega]^{nd}$ of the basic translation. Finally, we will define a "nonnested translation" $[\Omega]^{nn}$, obtained from $[\Omega]^{nd}$ by eliminating nested expressions in the bodies of rules in favor of additional atoms. The nonnested translation is a conservative extension of Ω, in the sense that dropping the new atoms from its answer sets gives the answer sets for Ω.

The translations defined in this paper can be of interest for several reasons. First, the definition of an answer set for programs with weight constraints is technically somewhat complicated. Instead of introducing that definition, we can treat any program Ω with weight constraints as shorthand for its translation $[\Omega]$.

Second, the definition of program completion from (Clark 1978) has been extended to nondisjunctive programs with nested expressions (Lloyd and Topor 1984), and this extension is known to be equivalent to the definition of an answer set whenever the program is "tight" (Erdem and Lifschitz 2003). In view of this fact, answer sets for a tight logic program can be generated by running a satisfiability solver

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

on the program's completion (Babovich *et al.* 2000). Consequently, answer sets for a program Ω with weight constraints can be computed by running a satisfiability solver on the completion of one of the translations $[\Omega]^{nd}$, $[\Omega]^{nn}$, if that translation is tight. This idea has led to the creation of a new software system for computing answer sets, called CMODELS[2]; see (Erdem and Lifschitz 2003, Section 7) for details.

Third, recent work on the theory of logic programs with nested expressions has led to a simple theory of equivalent transformations of such programs. Two programs are said to be *weakly equivalent* if they have the same answer sets, and *strongly equivalent* if they remain weakly equivalent after adding an arbitrary set of rules to both of them. For instance, rule (2) is strongly equivalent to rule (1), so that replacing one of these rules by the other in any program does not affect that program's answer sets. The study of strong equivalence is important because we learn from it how one can simplify a part of a program without looking at the other parts. The main theorem of (Lifschitz *et al.* 2001) shows that the strong equivalence of programs with nested expressions is characterized by the truth tables of the three-valued logic known as the logic of here-and-there.[3] Our translations can be used to prove the strong equivalence of programs with weight constraints using that logic.

The possibility of translating programs with cardinality constraints into the language of nonnested programs at the price of introducing new atoms was first established by Marek and Remmel [(2002)]. Our nonnested translation is more general, because it is applicable to programs with arbitrary weight constraints. Its other advantage is that, in the special case when all weights in the program are expressed by integers of a limited size (in particular, in the case of cardinality constraints) it does not make the program exponentially bigger. (In the translation from (Marek and Remmel 2002, Section 3), the number of rules introduced in part (II) can be exponentially large.)[4]

We begin with a review of programs with nested expressions (Section 2) and programs with weight constraints (Section 3). The translations are defined in Section 4, and their use for proving strong equivalence of programs with weight constraints is discussed in Section 5. Proofs of more difficult theorems are relegated to Section 6. Some properties of programs with nested expressions proved in that section, such as the completion lemma and the lemma on explicit definitions, may be of more general interest.

[2] `http://www.cs.utexas.edu/users/tag/cmodels.html` .

[3] The close relationship between answer sets and the logic of here-and-there was first discovered by Pearce [(1997)].

[4] The use of additional atoms to keep the program small in the process of eliminating nested expressions is discussed in (Pearce *et al.* 2002). In case of the transition from $[\Omega]^{nd}$ to $[\Omega]^{nn}$, the role of additional atoms is even more significant: both the basic and nondisjunctive translations can be exponentially bigger than the original program, and the use of new atoms allows us to scale $[\Omega]^{nd}$ back down approximately to the size of Ω. The other reason why we are not applying here the translation from (Pearce *et al.* 2002) to $[\Omega]^{nd}$ is that it would not give a nondisjunctive program.

2 Programs with Nested Expressions

2.1 Syntax

A *literal* is a propositional atom possibly prefixed with the classical negation sign $\neg$. *Elementary formulas* are literals and the symbols $\bot$ ("false") and $\top$ ("true"). *Formulas* are built from elementary formulas using the unary connective *not* (negation as failure) and the binary connectives , (conjunction) and ; (disjunction). A *rule with nested expressions* has the form

$$Head \leftarrow Body \tag{4}$$

where both *Body* and *Head* are formulas. For instance, (1) is a formula; it can be used as shorthand for the rule

$$a; not\ a \leftarrow \top.$$

The expression

$$\neg a \leftarrow not\ a \tag{5}$$

is an example of a rule containing classical negation.

A *program with nested expressions* is any set of rules with nested expressions.

2.2 Semantics

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

- for a literal l, $Z \models l$ if $l \in Z$
- $Z \models \top$
- $Z \not\models \bot$
- $Z \models (F, G)$ if $Z \models F$ and $Z \models G$
- $Z \models (F; G)$ if $Z \models F$ or $Z \models G$
- $Z \models not\ F$ if $Z \not\models F$.

We say that Z *satisfies* a program Π (symbolically, $Z \models \Pi$) if, for every rule (4) in Π, $Z \models Head$ whenever $Z \models\ Body$.

The *reduct*[5] F^Z of a formula F with respect to a consistent set Z of literals is defined recursively as follows:

- for elementary F, $F^Z = F$
- $(F, G)^Z = F^Z, G^Z$
- $(F; G)^Z = F^Z; G^Z$
- $(not\ F)^Z = \begin{cases} \bot\ , & \text{if } Z \models F, \\ \top\ , & \text{otherwise} \end{cases}$

[5] This definition of reduct is the same as the one in (Lifschitz *et al.* 2001), except that the condition $Z \models F^Z$ is replaced with $Z \models F$. It is easy to check by structural induction that the two conditions are equivalent.

The *reduct* Π^Z of a program Π with respect to Z is the set of rules

$$\mathit{Head}^Z \leftarrow \mathit{Body}^Z$$

for each rule (4) in Π. For instance, the reduct of (2) with respect to Z is

$$a \leftarrow \top \tag{6}$$

if $a \in Z$, and

$$a \leftarrow \bot \tag{7}$$

otherwise.

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

For instance, the reduct of (2) with respect to $\{a\}$ is (6), and $\{a\}$ is a minimal set satisfying (6); consequently, $\{a\}$ is an answer set for (2). On the other hand, the reduct of (2) with respect to $\emptyset$ is (7), and $\emptyset$ is a minimal set satisfying (7); consequently, $\emptyset$ is an answer set for (2) as well.

2.3 A Useful Abbreviation

The following abbreviation is used in the definition of the translation $[\Omega]$ in Section 4. For any formulas $F_1, \ldots, F_n$ and any set X of subsets of $\{1, \ldots, n\}$, by

$$\langle F_1, \ldots, F_n \rangle : X$$

we denote the formula

$$\mathop{;}_{I \in X} \big(\mathop{,}_{i \in I} F_i \big). \tag{8}$$

The use of the "big comma" and the "big semicolon" in (8) to represent a multiple conjunction and a multiple disjunction is similar to the familiar use of $\bigwedge$ and $\bigvee$. In particular, the empty conjunction is understood as $\top$, and the empty disjunction as $\bot$.

For instance, if X is the set of all subsets of $\{1, \ldots, n\}$ of cardinality ≥ 3, then (8) expresses, intuitively, that at least 3 of the formulas $F_1, \ldots, F_n$ are true. It is easy to check, for this X, that a consistent set Z of literals satisfies (8) iff Z satisfies at least 3 of the formulas $F_1, \ldots, F_n$. This observation can be generalized:

Proposition 1

Assume that for every subset I of $\{1, \ldots, n\}$ that belongs to X, all supersets of I belong to X also. For any formulas $F_1, \ldots, F_n$ and any consistent set Z of literals,

$$Z \models \langle F_1, \ldots, F_n \rangle : X \text{ iff } \{i : Z \models F_i\} \in X.$$

Proof

$$
\begin{array}{lll}
Z \models \langle F_1, \dots, F_n \rangle : X & \text{iff} & \text{for some } I \in X, \text{ for all } i, \text{ if } i \in I \text{ then } Z \models F_i \\
& \text{iff} & \text{for some } I \in X,\ I \subseteq \{i : Z \models F_i\} \\
& \text{iff} & \text{for some } I \in X,\ I = \{i : Z \models F_i\} \\
& \text{iff} & \{i : Z \models F_i\} \in X. \quad \square
\end{array}
$$

2.4 Strong Equivalence

Recall that programs Π_1 and Π_2 are said to be *strongly equivalent* to each other if, for every program Π, the union $\Pi_1 \cup \Pi$ has the same answer sets as $\Pi_2 \cup \Pi$. This concept is essential both for applications of our translations and for the proof of their soundness.

The method of proving strong equivalence proposed in (Lifschitz *et al.* 2001) is particularly simple for programs that do not contain classical negation. We first rewrite both programs in the syntax of propositional formulas by writing every rule (4) as the implication $Body \rightarrow Head$ and replacing every comma in the rule with $\wedge$, every semicolon with $\vee$, and every occurrence of negation as failure with $\neg$. For instance, rule (2) in this notation is

$$\neg\neg a \rightarrow a.$$

Then we check whether the rules of each of the programs Π_1, Π_2 are entailed by the rules of the other in the logic of here-and-there; if they are, then Π_1 and Π_2 are strongly equivalent to each other, and the other way around ((Lifschitz *et al.* 2001), Theorem 1).

The logic of here-and-there was originally defined in (Heyting 1930). Its definition and basic properties are discussed in (Lifschitz *et al.* 2001, Section 2). It is a three-valued logic, intermediate between intuitionistic and classical. Recall that a natural deduction system for intuitionistic logic can be obtained from the corresponding classical system (Bibel and Eder 1993, Table 3) by dropping the law of the excluded middle

$$F \vee \neg F$$

from the list of postulates. The logic of here-and-there, on the other hand, is the result of replacing the excluded middle in the classical system with the weaker axiom schema

$$F \vee (F \rightarrow G) \vee \neg G.$$

In addition to all intuitionistically provable formulas, the set of theorems of the logic of here-and-there includes, for instance, the weak law of the excluded middle

$$\neg F \vee \neg\neg F$$

and De Morgan's law

$$\neg(F \wedge G) \leftrightarrow \neg F \vee \neg G$$

(the dual law can be proved even intuitionistically).

As an example of the use of this idea, note that the absorption laws

$$a \vee (a \wedge b) \leftrightarrow a$$
$$a \wedge (a \vee b) \leftrightarrow a$$

are provable in the logic of here-and-there (their usual proofs in the natural deduction formalization of propositional logic do not use the law of the excluded middle). It follows that, in any program, $a; (a, b)$ and $a, (a; b)$ can be replaced by a without changing the program's answer sets. In particular, if we take a program containing a multiple disjunction of the form (8) and restrict this disjunction to the sets I that are minimal in X, then the answer sets of the program will remain the same.

As another example, let us verify that (1) is strongly equivalent to (2) by proving the equivalence

$$a \vee \neg\, a \leftrightarrow \neg\neg a \rightarrow a$$

in the logic of here-and-there. The proof left-to-right is straightforward, by considering the cases a, $\neg a$. Right-to-left, use the instance $\neg a \vee \neg\neg a$ of the weak law of the excluded middle and consider the cases $\neg a$, $\neg\neg a$.

The extension of this method to programs with classical negation is based on the fact that classical negation can be eliminated from any program Π by a simple syntactic transformation. For every atom a that occurs in Π after the classical negation symbol $\neg$, choose a new atom a' and replace all occurrences of $\neg a$ with a'. The answer sets for the resulting program Π' that do not contain any of the pairs $\{a, a'\}$ are in a 1–1 correspondence with the answer sets for Π (Lifschitz *et al.* 2001, Section 5). If the rules of each of the programs Π'_1, Π'_2 can be derived from the rules of the other program and the formulas $\neg(a \wedge a')$ in the logic of here-and-there then Π_1 and Π_2 are strongly equivalent to each other, and the other way around ((Lifschitz *et al.* 2001), Theorem 2).

3 Programs with Weight Constraints

3.1 Syntax

A *rule element* is a literal (*positive* rule element) or a literal prefixed with *not* (*negative* rule element). A *weight constraint* is an expression of the form

$$L \leq \{c_1 = w_1, \ldots, c_m = w_m\} \leq U \tag{9}$$

where

- each of L, U is (a symbol for) a real number or one of the symbols $-\infty$, $+\infty$,
- $c_1, \ldots, c_m$ $(m \geq 0)$ are rule elements, and

- $w_1, \dots, w_m$ are nonnegative real numbers ("weights").[6]

The part $L \leq$ can be omitted if $L = -\infty$; the part $\leq U$ can be omitted if $U = +\infty$. A *rule with weight constraints* is an expression of the form

$$C_0 \leftarrow C_1, \dots, C_n \tag{10}$$

where $C_0, \dots, C_n$ $(n \geq 0)$ are weight constraints. We will call the rule elements of C_0 the *head elements* of rule (10).

Finally, a *program with weight constraints* is a set of rules with weight constraints.[7]

This syntax becomes a generalization of the basic syntax of logic programs for which the answer set semantics was originally defined (Gelfond and Lifschitz 1988) if we identify a rule element c with the weight constraint

$$1 \leq \{c = 1\}.$$

By

$$\leftarrow C_1, \dots, C_n$$

we denote the rule

$$1 \leq \{\ \} \leftarrow C_1, \dots, C_n.$$

A *cardinality constraint* is a weight constraint with all weights equal to 1. A cardinality constraint

$$L \leq \{c_1 = 1, \dots, c_m = 1\} \leq U$$

can be abbreviated as

$$L \leq \{c_1, \dots, c_m\} \leq U. \tag{11}$$

3.2 Semantics

The definition of an answer set for programs with weight constraints in (Simons *et al.* 2002) uses the following auxiliary definitions. A consistent set Z of literals *satisfies* a weight constraint (9) if the sum of the weights w_j for all j such that $Z \models c_j$ is not less than L and not greater than U. For instance, Z satisfies cardinality constraint (3) iff Z contains at most one of the atoms a, b. About a program Ω with weight constraints we say that Z *satisfies* Ω if, for every rule (10) in Ω, Z satisfies C_0 whenever Z satisfies $C_1, \dots, C_n$. As in the case of nested expressions,

[6] In (Simons *et al.* 2002), weights are not required to be nonnegative, and the meaning of a program with negative weights is defined by describing a method for eliminating them. Unfortunately, this preprocessing step leads to some results that seem unintuitive. For instance, it turns out that the one-rule programs

$$1 \leq \{p = 1\} \leftarrow 0 \leq \{p = 2, p = -1\}$$

and

$$1 \leq \{p = 1\} \leftarrow 0 \leq \{p = 1\}$$

have different answer sets.

[7] In (Simons *et al.* 2002), programs are not allowed to contain classical negation. But classical negation is allowed in the input files of the current version of SMODELS.

we will use $\models$ to denote the satisfaction relation for both weight constraints and programs with weight constraints.

The next part of the semantics of weight constraints is the definition of the reduct for weight constraints of the form

$$L \leq \{c_1 = w_1, \ldots, c_m = w_m\}.$$

The *reduct* $(L \leq S)^Z$ of a weight constraint $L \leq S$ with respect to a consistent set Z of literals is the weight constraint $L^Z \leq S'$, where

- S' is obtained from S by dropping all pairs $c = w$ such that c is negative, and
- L^Z is L minus the sum of the weights w for all pairs $c = w$ in S such that c is negative and $Z \models c$.

For instance, the reduct of the constraint

$$1 \leq \{not\ a = 3, not\ b = 2\}$$

relative to $\{a\}$ is

$$-1 \leq \{\ \}.$$

The *reduct* of a rule

$$L_0 \leq S_0 \leq U_0 \leftarrow L_1 \leq S_1 \leq U_1, \ldots, L_n \leq S_n \leq U_n \tag{12}$$

with respect to a consistent set Z of literals is

- the set of rules of the form

  $$l \leftarrow (L_1 \leq S_1)^Z, \ldots, (L_n \leq S_n)^Z$$

  where l is a positive head element of (12) such that $Z \models l$, if, for every i $(1 \leq i \leq n)$, $Z \models S_i \leq U_i$;
- the empty set, otherwise.

The *reduct* Ω^Z of a program Ω with respect to Z is the union of the reducts of the rules of Ω.

Consider, for example, the one-rule program

$$1 \leq \{a = 2\} \leq 2 \leftarrow 1 \leq \{not\ a = 3, not\ b = 2\} \leq 4. \tag{13}$$

Since the only head element of (13) is a, the reduct of this rule with respect to a set Z of atoms is empty if $a \notin Z$. Consider the case when $a \in Z$. Since

$$Z \models \{not\ a = 3, not\ b = 2\} \leq 4,$$

the reduct consists of one rule

$$a \leftarrow (1 \leq \{not\ a = 3, not\ b = 2\})^Z.$$

It is clear from the definition of the reduct of a program above that every rule in a reduct satisfies two conditions:

- its head is a literal, and

* every member of its body has the form $L \leq S$ where S does not contain negative rule elements.

A rule satisfying these conditions is called a *Horn rule*. If a program Ω consists of Horn rules then there is a unique minimal set Z of literals such that $Z \models \Omega$. This set is called the *deductive closure* of Ω and denoted by $cl(\Omega)$.

Finally, a consistent set Z of literals is an *answer set* for a program Ω if $Z \models \Omega$ and $cl(\Omega^Z) = Z$.

To illustrate this definition, assume that Ω is (3). Set $\{a, b\}$ is not an answer set for Ω because it does not satisfy Ω. Let us check that every proper subset of $\{a, b\}$ is an answer set. Clearly, every such subset satisfies Ω. It remains to show that each of these sets is the deductive closure of the corresponding reduct of Ω.

* $\Omega^{\emptyset}$ is empty, so that $cl(\Omega^{\emptyset}) = \emptyset$.
* $\Omega^{\{a\}}$ consists of the single rule a, so that $cl(\Omega^{\{a\}}) = \{a\}$.
* $\Omega^{\{b\}}$ consists of the single rule b, so that $cl(\Omega^{\{b\}}) = \{b\}$.

To give another example, let Ω be (13). Set $\{b\}$ is not an answer set for Ω because it does not satisfy Ω. The other subsets of $\{a, b\}$ satisfy Ω. Consider the corresponding reducts.

* $\Omega^{\emptyset}$ is empty, so that $cl(\Omega^{\emptyset}) = \emptyset$.
* $\Omega^{\{a\}}$ is
$$a \leftarrow -1 \leq \{\}.$$
Consequently, $cl(\Omega^{\{a\}}) = \{a\}$.
* $\Omega^{\{a,b\}}$ is
$$a \leftarrow 1 \leq \{\}$$
Consequently, $cl(\Omega^{\{a,b\}}) = \emptyset \neq \{a, b\}$.

We conclude that the answer sets for (13) are $\emptyset$ and $\{a\}$.

4 Translations

4.1 Basic Translation

In this section, we give the main definition of this paper — the description of a translation from the language of weight constraints to the language of nested expressions — and state a theorem about the soundness of this translation. The definition of the translation consists of 4 parts.

1. The translation of a constraint of the form

$$L \leq \{c_1 = w_1, \dots, c_m = w_m\} \tag{14}$$

is the nested expression

$$\langle c_1, \dots, c_m \rangle : \left\{ I : L \leq \textstyle\sum_{i \in I} w_i \right\} \tag{15}$$

where I ranges over the subsets of $\{1, \dots, m\}$. We denote the translation of $L \leq S$ by $[L \leq S]$.

2. The translation of a constraint of the form

$$\{c_1 = w_1, \dots, c_m = w_m\} \leq U \tag{16}$$

is the nested expression

$$not\ \left(\langle c_1, \dots, c_m \rangle : \left\{I : U < \textstyle\sum_{i \in I} w_i\right\}\right). \tag{17}$$

where I ranges over the subsets of $\{1, \dots, m\}$. We denote the translation of $S \leq U$ by $[S \leq U]$.

3. The translation of a general weight constraint is defined by

$$[L \leq S \leq U] = [L \leq S], [S \leq U].$$

Recall that $L \leq S$ is shorthand for $L \leq S \leq \infty$, and $S \leq U$ is shorthand for $-\infty \leq S \leq U$; translations of weight constraints of these special types have been defined earlier. It is easy to see that the old definition of $[L \leq S]$ gives a nested expression equivalent to $[L \leq S \leq \infty]$, and similarly for $[S \leq U]$.

4. For any program Ω with weight constraints, its translation $[\Omega]$ is the program with nested expressions obtained from Ω by replacing each rule (10) with

$$(l_1; not\ l_1), \dots, (l_p; not\ l_p), [C_0] \leftarrow [C_1], \dots, [C_n] \tag{18}$$

where $l_1, \dots, l_p$ are the positive head elements of (10).

The conjunctive terms in $(l_1; not\ l_1), \dots, (l_p; not\ l_p)$ express, intuitively, that we are free to decide about every positive head element of the rule whether or not to include it in the answer set.

To illustrate this definition, let us apply it first to program (3). The translation of the cardinality constraint $0 \leq \{a, b\} \leq 1$ is

$$[0 \leq \{a, b\}], [\{a, b\} \leq 1]. \tag{19}$$

The first conjunctive term is

$$\langle a, b \rangle : \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$$

which equals

$$\top; a; b; (a, b)$$

and is equivalent to $\top$. Similarly, the second conjunctive term is equivalent to $not\ (a, b)$. Consequently, (19) can be written as $not\ (a, b)$. It follows that the translation of program (3) can be written as

$$(a; not\ a), (b; not\ b), not\ (a; b). \tag{20}$$

Similarly, we can check that program (13) turns into

$$a \leftarrow (not\ a; not\ b), not\ (not\ a, not\ b).$$

The translation defined above is sound:

Theorem 1
For any program Ω with weight constraints, Ω and $[\Omega]$ have the same answer sets.

We will conclude this section with a few comments about translating weight constraints of the forms $L \leq S$ and $S \leq U$.

In Section 3 we have agreed to identify any rule element c with the cardinality constraint $1 \leq \{c\}$, and to drop the head of a rule with weight constraints when this head is $1 \leq \{\ \}$. It is easy to check that $[1 \leq \{c\}]$ is equivalent to c, and $[1 \leq \{\ \}]$ is equivalent to $\bot$.

If the weights $w_1, \ldots, w_m$ are integers then the inequality in (17) is equivalent to $\lfloor U \rfloor + 1 \leq \sum_{i \in I} w_i$. Consequently, in the case of integer weights (in particular, in the case of cardinality constraints), $[S \leq U]$ can be written as *not* $[\lfloor U \rfloor + 1 \leq S]$. This is similar to a transformation that is used by the preprocessor LPARSE of system SMODELS.

The sign $<$ in place of $\leq$ is not allowed in weight constraints. But sometimes it is convenient to write expressions of the form

$$[L < \{c_1 = w_1, \ldots, c_m = w_m\}]$$

understood as shorthand for

$$\langle c_1, \ldots, c_m \rangle : \left\{ I : L < \textstyle\sum_{i \in I} w_i \right\}. \tag{21}$$

Using this notation, we can write $[S \leq U]$ as *not* $[U < S]$.

Finally, note that each of the sets X used in the expressions $\langle c_1, \ldots, c_m \rangle : X$ in formulas (15), (17) and (21) satisfies the assumption of Proposition 1 (Section 2.3), because the weights w_i are nonnegative.

4.2 Nondisjunctive Translation

A rule with nested expressions (Section 2) is *nondisjunctive* if its head is a literal or $\bot$. A *nondisjunctive program* is a program with nested expressions whose rules are nondisjunctive.

For any program Ω with weight constraints, its *nondisjunctive translation* $[\Omega]^{nd}$ is the nondisjunctive program obtained from Ω by replacing each rule (10) with $p+1$ rules

$$\begin{array}{ll} l_j \leftarrow not\ not\ l_j, [C_1], \ldots, [C_n] & \quad (1 \leq j \leq p), \\ \bot \leftarrow not\ [C_0], [C_1], \ldots, [C_n], & \end{array} \tag{22}$$

where $l_1, \ldots, l_p$ are the positive head elements of (10).

For example, if Π is (3) then $[\Pi]$, as we have seen, is (20); the nondisjunctive translation $[\Pi]^{nd}$ of the same program is

$$\begin{array}{l} a \leftarrow not\ not\ a, \\ b \leftarrow not\ not\ b, \\ \bot \leftarrow not\ not\ (a; b). \end{array} \tag{23}$$

Proposition 2
For any program Ω with weight constraints, $[\Omega]^{nd}$ is strongly equivalent to $[\Omega]$.

In combination with Theorem 1, this fact shows that the nondisjunctive translation is sound: Ω and $[\Omega]^{nd}$ have the same answer sets.

Its proof is based on the following well-known fact about intuitionistic logic:

Fact 1
If F is a propositional combination of formulas $F_1, \dots, F_m$ then $F \vee \neg F$ is intuitionistically derivable from $F_1 \vee \neg F_1, \dots, F_m \vee \neg F_m$.

Proof of Proposition 2
We will show that formula (18) is equivalent to the conjunction of the formulas (22) in the logic of here-and-there. By Fact 1, the formula

$$[C_0] \vee \neg[C_0] \tag{24}$$

is entailed by the formulas $c \vee \neg c$ for all head elements c of rule (10). For every negative c, $c \vee \neg c$ is provable in the logic of here-and-there. It follows that (24) is derivable in the logic of here-and-there from $l_1 \vee \neg l_1, \dots, l_p \vee \neg l_p$. Consequently, $\neg\neg[C_0] \equiv [C_0]$ is derivable from these formulas as well. Hence (18) is equivalent in the logic of here-and-there to the rule

$$(l_1; not\ l_1), \dots, (l_p; not\ l_p), not\ not\ [C_0] \leftarrow [C_1], \dots, [C_n]$$

which can be broken into the rules

$$\begin{array}{ll} l_j; not\ l_j \leftarrow [C_1], \dots, [C_n] & (1 \le j \le p), \\ not\ not\ [C_0] \leftarrow [C_1], \dots, [C_n]. & \end{array}$$

The first line is equivalent to the first line of (22) in the logic of here-and-there. The second line is intuitionistically equivalent to the second line of (22). □

4.3 Eliminating Nested Expressions

A nondisjunctive rule is *nonnested* if its body is a conjunction of literals, each possibly prefixed with *not*. A *nonnested program* is a program whose rules are nonnested. Thus the syntactic form of nonnested programs is the same as in the simple case reviewed at the beginning of Introduction, except that the head of a nonnested rule can be $\bot$, and that literals are allowed in place of atoms.

Since the answer sets for a nonnested program have the anti-chain property, turning a program with weight constraints into a nonnested program with the same answer sets is, generally, impossible. But we can turn any program with weight constraints into its nonnested conservative extension—into a program that may contain new atoms; dropping the new atoms from the answer sets of the translation gives the answer sets for the original program.

Each of the new atoms introduced in the nonnested translation $[\Omega]^{nn}$ below is, intuitively, an "abbreviation" for some formula related to the nondisjunctive translation $[\Omega]^{nd}$. For instance, to eliminate the nesting of negations from the first line of the nondisjunctive translation (22), we will introduce, for every j, a new atom $q_{not\ l_j}$, and replace that line with the rules

$$
\begin{aligned}
&q_{not\ l_j} \leftarrow not\ l_j, \\
&l_j \leftarrow not\ q_{not\ l_j}, [C_1], \ldots, [C_n]
\end{aligned}
$$

$(1 \leq j \leq p)$. The first of these rules tells us that the new atom $q_{not\ l_j}$ is used to "abbreviate" the formula $not\ l_j$. The second rule is the first of rules (22) with this subformula replaced by the corresponding atom. For instance, the nondisjunctive translation (23) of program (3) turns after this transformation into

$$
\begin{aligned}
&q_{not\ a} \leftarrow not\ a, \\
&a \leftarrow not q_{not\ a}, \\
&q_{not\ b} \leftarrow not\ b, \\
&b \leftarrow not\ q_{not\ b}, \\
&\bot \leftarrow not\ not\ (a; b).
\end{aligned}
\tag{25}
$$

Introducing the atoms $q_{not\ l_j}$ brings us very close to the goal of eliminating nesting altogether, because every rule of the program obtained from $[\Omega]^{nd}$ by this transformation is strongly equivalent to a set of nonnested rules. One way to eliminate nesting is to convert the body of every rule to a "disjunctive normal form" using De Morgan laws, the distributivity of conjunction over disjunction, and, in case of the second line of (22), double negation elimination.[8] After that, we can break every rule into several nonnnested rules, each corresponding to one of the disjunctive terms of the body. For instance, the last rule of (25) becomes

$$\bot \leftarrow a; b$$

after the first step and

$$
\begin{aligned}
&\bot \leftarrow a, \\
&\bot \leftarrow b
\end{aligned}
$$

after the second.

The definition of $[\Omega]^{nn}$ below follows a different approach to the elimination of the remaining nested expressions. Besides the "negation atoms" of the form $q_{not\ l_j}$, it introduces other new atoms, to make the translation of weight constraints more compact in some cases. These "weight atoms" have the forms $q_{w \leq S}$ and $q_{w<S}$, where w is a number and S is an expression of the form $\{c_1 = w_1, \ldots, c_m = w_m\}$ for some rule elements $c_1, \ldots, c_m$ and nonnegative numbers $w_1, \ldots, w_m$. They "abbreviate" the formulas $[w \leq S]$ and $[w < S]$ respectively.

In the following definition, $\{c_1 = w_1, \ldots, c_m = w_m\}'$, where $m > 0$, stands for $\{c_1 = w_1, \ldots, c_{m-1} = w_{m-1}\}$. Consider a nonnested program Π that may contain atoms of the forms $q_{w \leq S}$ and $q_{w<S}$. We say that Π is *closed* if

- for each atom of the form $q_{w \leq S}$ that occurs in Π, Π contains the rule

$$q_{w \leq S} \tag{26}$$

[8] Double negation elimination in the body of a rule with the head $\bot$ is intuitionistically valid.

if $w \le 0$, and the pair of rules

$$\begin{aligned} q_{w \le S} &\leftarrow q_{w \le S'}, \\ q_{w \le S} &\leftarrow c_m, q_{w-w_m \le S'} \end{aligned} \tag{27}$$

if $0 < w \le w_1 + \cdots + w_m$;

- for each atom of the form $q_{w<S}$ that occurs in Π, Π contains the rule

$$q_{w<S} \tag{28}$$

if $w < 0$, and the pair of rules

$$\begin{aligned} q_{w<S} &\leftarrow q_{w<S'}, \\ q_{w<S} &\leftarrow c_m, q_{w-w_m<S'} \end{aligned} \tag{29}$$

if $0 \le w < w_1 + \cdots + w_m$.

We define the nonnested translation $[L \le S \le U]^{nn}$ of a weight constraint $L \le S \le U$ as the conjunction

$$q_{L \le S}, \mathit{not}\ q_{U<S}.$$

Now we are ready to define the nonnested translation of a program. For any program Ω with weight constraints, $[\Omega]^{nn}$ is the smallest closed program that contains, for every rule

$$L_0 \le S_0 \le U_0 \leftarrow C_1, \ldots, C_n$$

of Ω, the rules

$$q_{\mathit{not}\ l} \leftarrow \mathit{not}\ l \tag{30}$$

and

$$l \leftarrow \mathit{not}\ q_{\mathit{not}\ l}, [C_1]^{nn}, \ldots, [C_n]^{nn} \tag{31}$$

for each of its positive head elements l, and the rules

$$\begin{aligned} \bot &\leftarrow \mathit{not}\ q_{L_0 \le S_0}, [C_1]^{nn}, \ldots, [C_n]^{nn}, \\ \bot &\leftarrow q_{U_0 < S_0}, [C_1]^{nn}, \ldots, [C_n]^{nn}. \end{aligned} \tag{32}$$

For instance, if Ω is (3) then rules (30)–(32) are

$$\begin{aligned} &q_{\mathit{not}\ a} \leftarrow \mathit{not}\ a, \\ &a \leftarrow \mathit{not}\ q_{\mathit{not}\ a}, \\ &q_{\mathit{not}\ b} \leftarrow \mathit{not}\ b, \\ &b \leftarrow \mathit{not}\ q_{\mathit{not}\ b}, \\ &\bot \leftarrow \mathit{not}\ q_{0 \le \{a,b\}}, \\ &\bot \leftarrow q_{1<\{a,b\}}. \end{aligned} \tag{33}$$

To make this program closed, we add to it the following "definitions" of the weight atoms $q_{0 \le \{a,b\}}$ and $q_{1<\{a,b\}}$, and, recursively, of the weight atoms that are used in these definitions:

$$
\begin{array}{l}
q_{0\leq\{a,b\}},\\
q_{1<\{a,b\}} \leftarrow q_{1<\{a\}},\\
q_{1<\{a,b\}} \leftarrow b, q_{0<\{a\}},\\
q_{0<\{a\}} \leftarrow q_{0<\{\}},\\
q_{0<\{a\}} \leftarrow a, q_{-1<\{\}},\\
q_{-1<\{\}}.
\end{array}
\tag{34}
$$

The nonnested translation of (3) consists of rules (33) and (34).

The following theorem describes the relationship between the answer sets for Ω and the answer sets for $[\Omega]^{nn}$. In the statement of the theorem, Q_Ω stands for the set of all new atoms that occur in $[\Omega]^{nn}$—both negation atoms $q_{not\ l}$ and weight atoms $w \leq S$, $w < S$.

Theorem 2

For any program Ω with weight constraints, $Z \mapsto Z \setminus Q_\Omega$ is a 1–1 correspondence between the answer sets for $[\Omega]^{nn}$ and the answer sets for Ω.

Recall that the introduction of the new atoms $q_{wL\leq S}$ and $q_{w<S}$ is motivated by the desire to make the translations of programs more compact. We will investigate now to what degree this goal has been achieved.

The basic translation $[C]$ of a weight constraint, as defined in Section 4.1, can be exponentially larger than C. For this reason, the basic and nondisjunctive translations of a program Ω are, generally, exponentially larger than Ω.

The nonnested translation of a program Ω consists of the rules (30)–(32) corresponding to all rules of Ω, and the additional rules (26)–(29) that make the program closed. The part consisting of rules (30)–(32) cannot be significantly larger than Ω, because each of the formulas $[C_i]^{nn}$ is short — it contains at most two atoms. The second part consists of the "definitions" of all weight atoms in $[\Omega]^{nn}$, and it contains at most two short rules for every such atom. Under what conditions can we guarantee that the number of weight atoms is not large in comparison with the size of Ω?

The *length* of a weight constraint (9) is m, and its *weight* is $w_1 + \cdots + w_m$. We will denote the length of C by $L(C)$, and the weight of C by $W(C)$.

Proposition 3

For programs Ω without non-integer weights, the number of weight atoms occurring in $[\Omega]^{nn}$ is $O\big(\sum L(C) \cdot W(C)\big)$, where the sum extends over all weight constraints C occurring in Ω.

If the weights in Ω come from a fixed finite set of integers (for instance, if every weight constraint in Ω is a cardinality constraint) then $W(C) = O(L(C))$, and the proposition above shows that the number of weight atoms in $[\Omega]^{nn}$ is not large in comparison with the size of Ω. Consequently, in this case $[\Omega]^{nn}$ cannot be large in comparison with Ω either.

Proof of Proposition 3
Let Ω be a program without non-integer weights. About a rule from $[\Omega]^{nn}$ we will say that it is *relevant* if for every weight atom $w \leq S$ or $w < S$ occurring in that rule there is a weight constraint (9) in Ω such that S is $\{c_1 = w_1, \dots, c_j = w_j\}$ for some $j \in \{0, \dots, m\}$, and

$$w \in \{-\max(w_1, \dots, w_m), \dots, w_1 + \dots + w_m\} \cup \{L, U\}.$$

It is clear that the number of weight atoms occurring in relevant rules can be estimated as $O\big(\sum L(C) \cdot W(C)\big)$. On the other hand, it is easy to see that the set of relevant rules contains the rules (30)–(32) corresponding to all rules of Ω, and that it is closed. Consequently, all rules in $[\Omega]^{nn}$ are relevant. $\square$

Without the assumption that all weights in Ω are integers, we can guarantee that the number of weight atoms occurring in $[\Omega]^{nn}$ is $O\big(\sum 2^{L(C)}\big)$.

5 Proving Strong Equivalence of Programs with Weight Constraints

For programs with weight constraints, the definition of strong equivalence is similar to the definition given above: Ω_1 and Ω_2 are *strongly equivalent* to each other if, for every program Ω with weight constraints, the union $\Omega_1 \cup \Omega$ has the same answer sets as $\Omega_2 \cup \Omega$. The method of proving strong equivalence of programs with weight constraints discussed in this section is based on the following proposition:

Proposition 4
Ω_1 is strongly equivalent to Ω_2 iff $[\Omega_1]$ is strongly equivalent to $[\Omega_2]$.

Proof
Assume that $[\Omega_1]$ is strongly equivalent to $[\Omega_2]$. Then, for any program with weight constraints Ω, $[\Omega_1] \cup [\Omega]$ has the same answer sets as $[\Omega_2] \cup [\Omega]$. The first program equals $[\Omega_1 \cup \Omega]$, and, by Theorem 1, has the same answer sets as $\Omega_1 \cup \Omega$. Similarly, the second program has the same answer sets as $\Omega_2 \cup \Omega$. Consequently Ω_1 is strongly equivalent to Ω_2.

Assume now that $[\Omega_1]$ is not strongly equivalent to $[\Omega_2]$. Consider the corresponding programs $[\Omega_1]'$, $[\Omega_2]'$ without classical negation, formed as described at the end of Section 2.4, and let *Cons* be the set of formulas $\neg(a \wedge a')$ for all new atoms a' occurring in these programs. By Theorem 2 from (Lifschitz *et al.* 2001), $[\Omega_1]' \cup \mathit{Cons}$ is not equivalent to $[\Omega_2]' \cup \mathit{Cons}$ in the logic of here-and-there. It follows by Theorem 1 from (Lifschitz *et al.* 2001) that there exists a unary program Π such that $[\Omega_1]' \cup \mathit{Cons} \cup \Pi$ and $[\Omega_2]' \cup \mathit{Cons} \cup \Pi$ have different collections of answer sets. (A program with nested expressions is said to be unary if each of its rules is an atom or has the form $a_1 \leftarrow a_2$ where a_1, a_2 are atoms.) Let Π^* be the program obtained from Π by replacing each atom of the form a' by $\neg a$. In view of the convention about identifying any literal l with the weight constraint $1 \leq \{l = 1\}$ (Section 3.1), Π^* can be viewed as a program with weight constraints, and it's easy to check that $[\Pi^*]'$ is strongly equivalent to Π. Then, for $i = 1, 2$, the program $[\Omega_i]' \cup \mathit{Cons} \cup \Pi$ has the same answer sets as the program $[\Omega_i]' \cup \mathit{Cons} \cup [\Pi^*]'$, which can be rewritten as $[\Omega_i \cup \Pi^*]' \cup \mathit{Cons}$. By the choice of Π, it follows that the collection of answer sets of $[\Omega_1 \cup \Pi^*]' \cup \mathit{Cons}$ is different from the collection of answer sets of $[\Omega_2 \cup \Pi^*]' \cup \mathit{Cons}$. Consequently, the same can be said about the pair of programs $[\Omega_1 \cup \Pi^*]$ and $[\Omega_2 \cup \Pi^*]$, and, by Theorem 1, about $\Omega_1 \cup \Pi^*$ and $\Omega_2 \cup \Pi^*$. It follows that Ω_1 is not strongly equivalent to Ω_2. $\square$

As an example, let us check that the program

$$\begin{array}{l} 1 \leq \{p, q\} \leq 1 \\ p \end{array} \tag{35}$$

is strongly equivalent to

$$\begin{array}{l} \leftarrow q \\ p. \end{array} \tag{36}$$

Rules (35), translated into the language of nested expressions and written in the syntax of propositional formulas, become

$$\begin{array}{l} (p \vee \neg p) \wedge (q \vee \neg q) \wedge (p \vee q) \wedge \neg(p \wedge q) \\ (p \vee \neg p) \wedge p. \end{array}$$

Rules (36), rewritten in a similar way, become

$$\begin{array}{l} \neg q \\ (p \vee \neg p) \wedge p. \end{array}$$

It is clear that each of these sets of formulas is intuitionistically equivalent to $\{p, \neg q\}$.

The fact that programs (35) and (36) are strongly equivalent to each other can be also proved directly, using the definition of strong equivalence and the definition of an answer set for programs with weight constraints. But this proof would not be as easy as the one above. Generally, to establish that a program Ω_1 is strongly equivalent to a program Ω_2, we need to show that for every program Ω and every consistent set Z of literals,

(a$_1$) $Z \models \Omega_1 \cup \Omega$ and
(b$_1$) $cl((\Omega_1 \cup \Omega)^Z) = Z$

if and only if

(a$_2$) $Z \models \Omega_2 \cup \Omega$ and
(b$_2$) $cl((\Omega_2 \cup \Omega)^Z) = Z$.

Sometimes we may be able to check separately that (a$_1$) is equivalent to (a$_2$) and that (b$_1$) is equivalent to (b$_2$), but in other cases this may not work. For instance, if Ω_1 is (35) and Ω_2 is (36) then (b$_1$) may not be equivalent to (b$_2$).

An alternative method of establishing the strong equivalence of programs with weight constraints is proposed in (Turner 2003, Section 6). According to that approach, we check that for every consistent set Z of literals and every subset Z' of Z,

(a$_3$) $Z \models \Omega_1$ and

(b$_3$) $Z' \models \Omega_1^Z$

if and only if

(a$_4$) $Z \models \Omega_2$ and
(b$_4$) $Z' \models \Omega_2^Z$.

6 Proofs of Theorems

6.1 Proof of Theorem 1

Lemma 1
For any weight constraint C and any consistent set Z of literals, $Z \models [C]$ iff $Z \models C$.

Proof
It is sufficient to prove the assertion of the lemma for constraints of the forms $L \leq S$ and $S \leq U$. Let S be $\{c_1 = w_1, \ldots, c_m = w_m\}$. Then, by Proposition 1 (Section 2.3),

$$\begin{aligned} Z \models [L \leq S] \quad & \text{iff} \quad \{i : Z \models c_i\} \in \{I : L \leq \textstyle\sum_{i \in I} w_i\} \\ & \text{iff} \quad L \leq \textstyle\sum_{i : Z \models c_i} w_i \\ & \text{iff} \quad Z \models L \leq S. \end{aligned}$$

Similarly,

$$\begin{aligned} Z \models [S \leq U] \quad & \text{iff} \quad \{i : Z \models c_i\} \notin \{I : U < \textstyle\sum_{i \in I} w_i\} \\ & \text{iff} \quad U \geq \textstyle\sum_{i : Z \models c_i} w_i \\ & \text{iff} \quad Z \models S \leq U. \quad \square \end{aligned}$$

Lemma 2
For any constraint $L \leq S$ and any consistent sets Z, Z' of literals,

$$Z' \models [L \leq S]^Z \text{ iff } Z' \models (L \leq S)^Z.$$

Proof
Let S be $\{c_1 = w_1, \ldots, c_m = w_m\}$ and let I stand for $\{1, \ldots, m\}$. It is immediate from the definition of the reduct in Section 2.2 that

$$\big(\langle F_1, \ldots, F_n \rangle : X\big)^Z = \langle F_1^Z, \ldots, F_n^Z \rangle : X. \tag{37}$$

For any subset J of I, let ΣJ stand for $\sum_{i \in J} w_i$. Using (37) and Proposition 1, we can rewrite the left-hand side of the equivalence to be proved as follows:

$$\begin{aligned} Z' \models [L \leq S]^Z \quad & \text{iff} \quad Z' \models \langle c_1^Z, \ldots, c_m^Z \rangle : \{J \subseteq I : L \leq \Sigma J\} \\ & \text{iff} \quad \{i \in I : Z' \models c_i^Z\} \in \{J \subseteq I : L \leq \Sigma J\} \\ & \text{iff} \quad L \leq \Sigma\{i \in I : Z' \models c_i^Z\} \end{aligned}$$

Let I' be the set of all $i \in I$ such that the rule element c_i is positive, and let I'' be the set of all $i \in I \setminus I'$ such that $Z \models c_i$. It is clear that c_i^Z is c_i for $i \in I'$, $\top$ for $i \in I''$, and $\bot$ for all other values of i. Consequently

$$\begin{array}{rll} Z' \models [L \leq S]^Z & \text{iff} & L \leq \Sigma\{i \in I' : Z' \models c_i\} + \Sigma I'' \\ & \text{iff} & L - \Sigma I'' \leq \Sigma\{i \in I' : Z' \models c_i\} \\ & \text{iff} & Z' \models (L^Z \leq S') \end{array}$$

where L^Z and S' are defined as in Section 3.2. It remains to notice that $(L \leq S)^Z = (L^Z \leq S')$. □

Lemma 3
For any constraint $S \leq U$ and any consistent set Z of literals,

$$[S \leq U]^Z = \begin{cases} \top\,, & \text{if } Z \models (S \leq U), \\ \bot\,, & \text{otherwise.} \end{cases}$$

Proof
By the definition of the reduct in Section 2.2, $[S \leq U]^Z$ is

- $\top$, if $Z \not\models [U < S]$,
- $\bot$, otherwise.

It remains to notice that $Z \not\models [U < S]$ iff $Z \models [S \leq U]$, and then iff $Z \models S \leq U$ by Lemma 1. □

In Lemmas 4–7, Ω is an arbitrary program with weight constraints. Recall that, according to Section 4.2, the nondisjunctive translation $[\Omega]^{nd}$ of Ω consists of rules of two kinds:

$$l_j \leftarrow not\ not\ l_j, [C_1], \ldots, [C_n] \tag{38}$$

and

$$\bot \leftarrow not\ [C_0], [C_1], \ldots, [C_n]. \tag{39}$$

We will denote the set of rules (38) corresponding to all rules of Ω by Π_1, and the set of rules (39) corresponding to all rules of Ω by Π_2, so that

$$[\Omega]^{nd} = \Pi_1 \cup \Pi_2. \tag{40}$$

Lemma 4
A consistent set Z of literals is an answer set for $[\Omega]^{nd}$ iff Z is an answer set for Π_1 and $Z \models \Pi_2$.

In view of (40), this is an instance of a general fact, proved in (Lifschitz *et al.* 1999) as Proposition 2, that can be restated as the following:

Fact 2
Let Π_1, Π_2 be programs with nested expressions such that the head of every rule in Π_2 is $\bot$. A consistent set Z of literals is an answer set for $\Pi_1 \cup \Pi_2$ iff Z is an answer set for Π_1 and $Z \models \Pi_2$.

Lemma 5
For any consistent set Z of literals, $Z \models \Omega$ iff $Z \models \Pi_2$.

Proof
It is sufficient to consider the case when Ω consists of a single rule (10). In this case, $Z \models \Omega$ iff

$$Z \models C_0 \text{ or, for some } i \ (1 \leq i \leq m),\ Z \not\models C_i.$$

On the other hand, $Z \models \Pi_2$ iff

$$Z \models [C_0] \text{ or, for some } i \ (1 \leq i \leq m),\ Z \not\models [C_i].$$

By Lemma 1, these conditions are equivalent to each other. $\square$

Lemma 6
For any consistent sets Z, Z' of literals, $Z' \models \Omega^Z$ iff $Z' \models \Pi_1^Z$.

Proof
It is sufficient to consider the case when Ω consists of a single rule (12). Then Π_1^Z consists of the rules

$$l \leftarrow (not\ not\ l)^Z, [L_1 \leq S_1]^Z, [S_1 \leq U_1]^Z, \ldots, [L_n \leq S_n]^Z, [S_n \leq U_n]^Z \tag{41}$$

for all positive head elements l of (12).

Case 1: for every i $(1 \leq i \leq n)$, $Z \models S_i \leq U_i$. Then, by Lemma 3, each of the formulas $[S_1 \leq U_1]^Z, \ldots, [S_n \leq U_n]^Z$ is $\top$. Note also that if $l \notin Z$ then $(not\ not\ l)^Z$ is $\bot$, so that (41) is satisfied by any consistent set of literals. Consequently Z' satisfies Π_1^Z iff, for each positive head element $l \in Z$,

$$Z' \models l \text{ or, for some } i \ (1 \leq i \leq m),\ Z' \not\models [L_i \leq S_i]^Z. \tag{42}$$

On the other hand, according to the definition of the reduct from Section 3.2, Ω^Z is the set of rules

$$l \leftarrow (L_1 \leq S_1)^Z, \ldots, (L_n \leq S_n)^Z$$

for all positive head elements l satisfied by Z. Then $Z' \models \Omega^Z$ iff, for each positive head element $l \in Z$,

$$Z' \models l \text{ or, for some } i \ (1 \leq i \leq m),\ Z' \not\models (L_i \leq S_i)^Z.$$

By Lemma 2, this condition is equivalent to (42).

Case 2: for some i, $Z \not\models S_i \leq U_i$. Then, by Lemma 3, one of the formulas $[S_i \leq U_i]^Z$ is $\bot$, so that each rule (41) is trivially satisfied by any Z'. On the other hand, in this case Ω^Z is empty. $\square$

Lemma 7
If set $cl(\Omega^Z)$ is consistent then it is the only answer set for Π_1^Z; otherwise, Π_1^Z has no answer sets.

Proof
Recall that $cl(\Omega^Z)$ is defined as the unique minimal set satisfying Ω^Z (Section 3.2). The answer sets for a program with nested expressions that does not contain negation as failure are defined as the minimal consistent sets satisfying that program (Section 2.2). It remains to notice that Ω^Z and Π_1^Z are satisfied by the same sets of literals (Lemma 6). □

Theorem 1
For any program Ω with weight constraints, Ω and $[\Omega]$ have the same answer sets.

Proof
By the definition of an answer set for programs with weight constraints (Section 3), a consistent set Z of literals is an answer set for Ω iff

$$cl(\Omega^Z) = Z \text{ and } Z \models \Omega.$$

By Lemmas 7 and 5, this is equivalent to the condition

$$Z \text{ is an answer set for } \Pi_1^Z \text{ and } Z \models \Pi_2.$$

By the definition of an answer set for programs with nested expressions (Section 2) and by Lemma 4, this is further equivalent to saying that Z is an answer set for $[\Omega]^{nd}$. By Proposition 2, $[\Omega]^{nd}$ has the same answer sets as $[\Omega]$. □

6.2 Two Lemmas on Programs with Nested Expressions

The idea of program completion (Clark 1978) is that the set of rules of a program with the same atom q in the head is the "if" part of a definition of q; the "only if" half of that definition is left implicit. If, for instance, the rule

$$q \leftarrow F$$

is the only rule in the program whose head is q then that rule is an abbreviated form of the assertion that q is equivalent to F.

Since in a rule with nested expressions the head is allowed to have the same syntactic structure as the body, the "only if" part of such an equivalence can be expressed by a rule also:

$$F \leftarrow q.$$

The lemma below shows that adding such rules to a program does not change its answer sets.

An occurrence of a formula F in a formula or a rule is *singular* if the symbol before this occurrence of F is $\neg$; otherwise the occurrence is *regular* (Lifschitz *et al.* 1999). The expression

$$F \leftrightarrow G$$

stands for the pair of rules

$$F \leftarrow G$$
$$G \leftarrow F.$$

Completion Lemma
Let Π be a program with nested expressions, and let Q be a set of atoms that do not have regular occurrences in the heads of the rules of Π. For every $q \in Q$, let $Def(q)$ be a formula. Then the program

$$\Pi \cup \{q \leftarrow Def(q) \ : \ q \in Q\}$$

has the same answer sets as the program

$$\Pi \cup \{q \leftrightarrow Def(q) \ : \ q \in Q\}.$$

In the special case when Q is a singleton this fact was first proved by Esra Erdem (personal communication).

In the statement of the completion lemma, if the atoms from Q occur neither in Π nor in the formulas $Def(q)$ then adding the rules $q \leftarrow Def(q)$ to Π extends the program by "explicit definitions" of "new" atoms. According to the lemma below, such an extension is conservative: the answer sets for Π can be obtained by dropping the new atoms from the answer sets for the extended program.

Lemma on Explicit Definitions
Let Π be a program with nested expressions, and let Q be a set of atoms that do not occur in Π. For every $q \in Q$, let $Def(q)$ be a formula that contains no atoms from Q. Then $Z \mapsto Z \setminus Q$ is a 1–1 correspondence between the answer sets for $\Pi \cup \{q \leftarrow Def(q) \ : \ q \in Q\}$ and the answer sets for Π.

The completion lemma and the lemma on explicit definitions can be proved as follows.

Lemma 8
Let Π be a program without negation as failure, and Z' a subset of a consistent set Z of literals. If the literals in $Z \setminus Z'$ do not have regular occurrences in the heads of the rules of Π and $Z \models \Pi$ then $Z' \models \Pi$.

The proof of this lemma uses the following fact that is easy to verify by structural induction:

Fact 3
Let F be a formula without negation as failure, Z a consistent set of literals and Z' a subset of Z. If $Z' \models F$ then $Z \models F$.

Proof of Lemma 8
Take a rule $Head \leftarrow Body$ in Π such that $Z' \models Body$. By Fact 3, $Z \models Body$, and consequently $Z \models Head$. Since the literals in $Z \setminus Z'$ do not have regular occurrences in $Head$, it follows that $Z' \models Head$. $\square$

Lemma 9
Let Π be a logic program, and let S be the set of literals that have regular occurrences in Π in the scope of negation as failure. For any pair Z_1, Z_2 of consistent sets of literals, if $Z_1 \cap S = Z_2 \cap S$ then $\Pi^{Z_1} = \Pi^{Z_2}$.

Proof

From the condition $Z_1 \cap S = Z_2 \cap S$ we conclude that for every formula F occurring in Π in the scope of negation as failure, $Z_1 \models F$ iff $Z_2 \models F$. Then the fact that $F^{Z_1} = F^{Z_2}$ for every formula F occurring in Π follows by structural induction. $\square$

Proof of the Completion Lemma

First consider the case when Π and the formulas $Def(q)$ do not contain negation as failure; the general case is discussed at the end of the proof. Let Π_1 stand for $\Pi \cup \{q \leftarrow Def(q) \ : \ q \in Q\}$, and Π_2 stand for $\Pi \cup \{q \leftrightarrow Def(q) \ : \ q \in Q\}$. We need to show that Z is minimal among the sets satisfying Π_1 iff Z is minimal among the sets satisfying Π_2.

Case 1: For every subset Z' of Z, if $Z' \models \Pi_1$ then $Z' \models \Pi_2$. The opposite holds also, because $\Pi_1 \subseteq \Pi_2$. Consequently, a subset of Z satisfies Π_1 iff it satisfies Π_2, which implies that Z is minimal among the sets satisfying Π_1 iff Z is minimal among the sets satisfying Π_2.

Case 2: For some subset Z' of Z, $Z' \models \Pi_1$ but $Z' \not\models \Pi_2$. Let Z'' be the intersection of all subsets X of Z such that

(i) $X \setminus Q = Z' \setminus Q$, and
(ii) for every $q \in Q$, if $X \models Def(q)$ then $q \in X$.

We will establish several properties of Z''. First,

$$Z'' \subseteq Z'. \tag{43}$$

Indeed, (i) holds for Z' as X; since Z' satisfies the program Π_1 that contains the rules $q \leftarrow Def(q)$, (ii) holds for Z' as well. Consequently, Z' is one of the sets X whose intersection we denoted by Z'', which implies (43).

Second, Z'' satisfies conditions (i) and (ii) as X, that is to say,

(i′) $Z'' \setminus Q = Z' \setminus Q$, and
(ii′) for every $q \in Q$, if $Z'' \models Def(q)$ then $q \in Z''$.

Property (i′) is a consequence of the fact that Z'' is the intersection of a nonempty family of sets X satisfying (i). To prove (ii′), take any $q \in Q$ such that $Z'' \models Def(q)$. Each superset of Z'' satisfies $Def(q)$ by Fact 3. Each set X that satisfies (i) and (ii) is a superset of Z'', so that each of these sets X contains q by (ii). As Z'' is the intersection of these sets, $q \in Z''$.

By (i′), all literals from $Z' \setminus Z''$ belong to Q, and consequently do not have regular occurences in the heads of the rules of Π. Since $Z' \models \Pi$, we can conclude by Lemma 8 that $Z'' \models \Pi$. By (ii′), Z'' satisfies the rules $q \leftarrow Def(q)$. Furthermore, Z'' satisfies each rule $Def(q) \leftarrow q$, because otherwise $Z'' \setminus \{q\}$ would have been a proper subset of Z'' that satisfies conditions (i) and (ii) as X, which is impossible by the choice of Z''. Consequently, $Z'' \models \Pi_2$. Since $Z' \not\models \Pi_2$, it follows that Z'' is a proper subset of Z'. Then Z'' is a proper subset of Z. Since Z has a proper subset satisfying Π_2, it is neither an answer set for Π_1 nor an answer set for Π_2. $\square$

We have proved the completion lemma for the case when Π and the formulas $Def(q)$ do not contain negation as failure. To prove the lemma in full generality, apply this special case to program Π^Z and the formulas $Def(q)^Z$.

Proof of the Lemma on Explicit Definitions
Denote the set of rules $q \leftarrow Def(q)$ for all $q \in Q$ by Δ. The assertion of the lemma can be divided into two parts, and we will prove them separately.

Claim 1: If Z is an answer set for $\Pi \cup \Delta$ then $Z \setminus Q$ is an answer set for Π.

Consider first the case when neither Π nor Δ contains negation as failure. Take an answer set Z for $\Pi \cup \Delta$ and a subset Z' of $Z \setminus Q$. Lemma 8 can be applied to program Δ and the subset $(Z \cap Q) \cup Z'$ of Z, because $Z \setminus ((Z \cap Q) \cup Z')$, as a part of $Z \setminus Q$, does not contain literals occurring in the heads of the rules of Δ. Consequently

$$(Z \cap Q) \cup Z' \models \Delta. \tag{44}$$

Since Z is an answer set for $\Pi \cup \Delta$,

$$(Z \cap Q) \cup Z' \models \Pi \cup \Delta \text{ iff } (Z \cap Q) \cup Z' = Z \text{ iff } Z' = Z \setminus Q.$$

Using (44), we conclude:

$$(Z \cap Q) \cup Z' \models \Pi \text{ iff } Z' = Z \setminus Q.$$

Since no element of Q occurs in Π, we can rewrite this as

$$Z' \models \Pi \text{ iff } Z' = Z \setminus Q.$$

Since Z' here is an arbitrary subset of $Z \setminus Q$, we proved that $Z \setminus Q$ is an answer set for Π.

To prove Claim 1 in the general case, consider an answer set Z for $\Pi \cup \Delta$. It is an answer set for $\Pi^Z \cup \Delta^Z$ also. By the special case of Claim 1 proved above, $Z \setminus Q$ is an answer set for Π^Z. Since no element of Q occurs in Π, $\Pi^{Z \setminus Q} = \Pi^Z$ (Lemma 9). It follows that $Z \setminus Q$ is an answer set for $\Pi^{Z \setminus Q}$, and consequently for Π.

Claim 2: If Z^* is an answer set for Π then there exists a unique answer set Z for $\Pi \cup \Delta$ such that $Z \setminus Q = Z^*$.

Consider first the case when neither Π nor Δ contains negation as failure. Let Z^* be an answer set for Π. Define

$$Z_0 = Z^* \cup \{q \in Q \ : \ Z^* \models Def(q)\}.$$

We will show that Z_0 is the only consistent set Z of literals with the properties from Claim 2. Clearly $Z_0 \setminus Q = Z^*$. We will check now that

(i) Z_0 satisfies $\Pi \cup \Delta$,
(ii) no proper subset of Z_0 satisfies $\Pi \cup \Delta$, and
(iii) every consistent set Z of literals that satisfies $\Pi \cup \Delta$ and has the property $Z \setminus Q = Z^*$ is a superset of Z_0.

To show that Z_0 satisfies Π, observe that Z^* satisfies Π and no element of Q occurs in Π. To show that Z_0 satisfies Δ, assume that $Z_0 \models Def(q)$. Since no element of

Q occurs in $Def(q)$, it follows that $Z^* \models Def(q)$, so that $q \in Z_0$. Assertion (i) is proved.

It is convenient to prove assertion (iii) next. Take a consistent set Z of literals that satisfies $\Pi \cup \Delta$ and has the property $Z \setminus Q = Z^*$. Since Z^* is an answer set for Π, and Π does not contain elements of Q, Z^* is disjoint from Q, so that

$$Z_0 \setminus Q = Z^* \setminus Q = Z^* = Z \setminus Q \subseteq Z. \tag{45}$$

Take any $q \in Z_0 \cap Q$. Since Z^* is disjoint from Q, q belongs to the second of the two sets whose union we denoted by Z_0, so that $Z^* \models Def(q)$. Since $Z^* = Z \setminus Q$ and the elements of Q do not occur in $Def(q)$, it follows that $Z \models Def(q)$. In view of the fact that Z satisfies Δ, we can conclude that $q \in Z$. Since q here is an arbitrary element of $Z_0 \cap Q$, we proved that $Z_0 \cap Q \subseteq Z$. In combination with (45), this fact shows that Z is a superset of Z_0.

To prove assertion (ii), assume that a proper subset Z of Z_0 satisfies $\Pi \cup \Delta$. Since the elements of Q do not occur in Π, it follows that $Z \setminus Q$ satisfies Π. On the other hand, $Z \setminus Q$ is a subset of Z^*. Since Z^* is an answer set for Π, it follows that $Z \setminus Q$ cannot be a proper subset of Z^*. Consequently $Z \setminus Q = Z^*$. Then, by assertion (iii), Z is a superset of Z_0, which is impossible, by the choice of Z.

To prove Claim 2 in the general case, consider an answer set Z^* for Π. It is an answer set for Π^{Z^*} also. By the special case of Claim 2 proved above, there exists a unique answer set Z for $\Pi^{Z^*} \cup \Delta^{Z^*}$ such that $Z \setminus Q = Z^*$. No element of Q occurs in Π or Δ in the scope of negation as failure. By Lemma 9 it follows that $\Pi^{Z^*} = \Pi^Z$ and $\Delta^{Z^*} = \Delta^Z$ for every Z such that $Z \setminus Q = Z^*$. Consequently, there exists a unique answer set Z for $\Pi^Z \cup \Delta^Z$ such that $Z \setminus Q = Z^*$. It follows that there exists a unique answer set Z for $\Pi \cup \Delta$ such that $Z \setminus Q = Z^*$. $\square$

6.3 Proof of Theorem 2

Let Ω be a program with weight constraints. Consider the subset Δ of its nonnested translation $[\Omega]^{nn}$ consisting of the rules whose heads are atoms from Q_Ω. The rules included in Δ have the forms (26)–(30); they "define" the atoms in Q_Ω. The rest of $[\Omega]^{nn}$ will be denoted by Π; the rules of Π have the forms (31) and (32). The union of these two programs is $[\Omega]^{nn}$:

$$[\Omega]^{nn} = \Pi \cup \Delta. \tag{46}$$

The idea of the proof of Theorem 2 is to transform $\Pi \cup \Delta$ into a program with the same answer sets so that Π will turn into $[\Omega]^{nd}$ and Δ will turn into a set of explicit definitions in the sense of Section 6.2, and then use the lemma on explicit definitions.

For every atom $q \in Q_\Omega$, define the formula $Def(q)$ as follows:

$$Def(q_{not\ l}) = not\ l$$

$$Def(q_{w \le S}) = \begin{cases} \top, & \text{if } w \le 0, \\ q_{w \le S'}; (c_m, q_{w-w_m \le S'}), & \text{if } 0 < w \le w_1 + \cdots + w_m, \\ \bot, & \text{otherwise} \end{cases}$$

$$Def(q_{w < S}) = \begin{cases} \top, & \text{if } w < 0, \\ q_{w < S'}; (c_m, q_{w-w_m < S'}), & \text{if } 0 \le w < w_1 + \cdots + w_m, \\ \bot, & \text{otherwise} \end{cases}$$

Lemma 10
Program $[\Omega]^{nn}$ has the same answer sets as

$$\Pi \cup \{q \leftrightarrow Def(q) \ : \ q \in Q_\Omega\}.$$

Proof
From the definitions of $[\Omega]^{nn}$ and Q_Ω we conclude that Δ consists of the following rules:

- rule (26) for every atom of the form $q_{w \le S}$ in Q_Ω such that $w \le 0$;
- rules (27) for every atom of the form $q_{w \le S} \in Q_\Omega$ such that

$$0 < w \le w_1 + \cdots + w_m;$$

- rule (28) for every atom of the form $q_{w < S}$ in Q_Ω such that $w < 0$;
- rules (29) for every atom of the form $q_{w < S}$ in Q_Ω such that

$$0 \le w < w_1 + \cdots + w_m;$$

- rule (30) for every atom of the form $q_{not\ l}$ in Q_Ω.

Consequently Δ is strongly equivalent to $\{q \leftarrow Def(q) \ : \ q \in Q_\Omega\}$. Then, by (46), program $[\Omega]^{nn}$ has the same answer sets as $\Pi \cup \{q \leftarrow Def(q) \ : \ q \in Q_\Omega\}$. The assertion to be proved follows by the completion lemma. □

Lemma 11
Let S be $\{c_1 = w_1, \ldots, c_m = w_m\}$. In the logic of here-and-there,

$$[w \le S] \leftrightarrow \begin{cases} \top, & \text{if } w \le 0, \\ [w \le S']; (c_m, [w - w_m \le S']), & \text{if } 0 < w \le w_1 + \cdots + w_m, \\ \bot, & \text{otherwise.} \end{cases}$$

$$[w < S] \leftrightarrow \begin{cases} \top, & \text{if } w < 0, \\ [w < S']; (c_m, [w - w_m < S']), & \text{if } 0 \le w < w_1 + \cdots + w_m, \\ \bot, & \text{otherwise.} \end{cases}$$

Proof

Recall that $[w \leq S]$ is an expression of the form (15), which stands for a disjunction of conjunctions (8). If $w \leq 0$ then the set after the : sign in (15) has the empty set as one of its elements, so that one of the disjunctive terms of this formula is the empty conjunction $\top$. If $w \geq w_1+\cdots+w_m$ then set after the : sign in (15) is empty, so that the formula is the empty disjunction $\bot$. Assume now that $0 < w_1 + \cdots + w_m \leq w$. Let I stand for $\{1, \ldots, m\}$ and let I' be $\{1, \ldots, m-1\}$. For any subset J of I, by ΣJ we denote the sum $\sum_{i \in J} w_i$. Then

$$\begin{aligned}
[w \leq S] \quad &= \mathop{;}\limits_{J \subseteq I \,:\, \Sigma J \geq w} \big(\mathop{,}\limits_{i \in J} c_i\big) \\
&\leftrightarrow \mathop{;}\limits_{J \subseteq I' \,:\, \Sigma J \geq w} \big(\mathop{,}\limits_{i \in J} c_i\big); \mathop{;}\limits_{J \subseteq I \,:\, m \in J, \Sigma J \geq w} \big(\mathop{,}\limits_{i \in J} c_i\big) \\
&= [w \leq S']; \mathop{;}\limits_{J \subseteq I' \,:\, \Sigma J + w_m \geq w} \big(\mathop{,}\limits_{i \in J \cup \{m\}} c_i\big) \\
&\leftrightarrow [w \leq S']; \big(c_m, \mathop{;}\limits_{J \subseteq I' \,:\, \Sigma J \geq w - w_m} \big(\mathop{,}\limits_{i \in J} c_i\big)\big) \\
&= [w \leq S']; (c_m, [(w - w_m) \leq S']).
\end{aligned}$$

The proof of the second equivalence is similar. $\square$

Lemma 12

Program

$$\{q \leftrightarrow Def(q) \;:\; q \in Q_\Omega\} \tag{47}$$

is strongly equivalent to

$$\begin{array}{l}
\{q_{not\ l} \leftrightarrow not\ l \;:\; q_{not\ l} \in Q_\Omega\} \cup \\
\{q_{w \leq S} \leftrightarrow [w \leq S] \;:\; q_{w \leq S} \in Q_\Omega\} \cup \\
\{q_{w < S} \leftrightarrow [w < S] \;:\; q_{w < S} \in Q_\Omega\}.
\end{array} \tag{48}$$

Proof

The rules of (48) can be obtained from the rules of (47) by replacing $Def(q_{w \leq S})$ with $[w \leq S]$ for the atoms $q_{w \leq S}$ in Q_Ω, and $Def(q_{w<S})$ with $[w < S]$ for the atoms $q_{w<S}$ in Q_Ω. Consequently, it is sufficient to show that, for every atom of the form $q_{w \leq S}$ in Q_Ω, the equivalences

$$Def(q_{w \leq S}) \leftrightarrow [w \leq S] \tag{49}$$

are derivable in the logic of here-and-there both from (47) and from (48), and similarly for atoms of the form $q_{w<S}$. The proofs for atoms of both kinds are similar, and we will only consider $q_{w \leq S}$. Let S be $\{c_1 = w_1, \ldots, c_m = w_m\}$.

The definition of $Def(q_{w \leq S})$ and the statement of Lemma 11 show that the right-hand side of (49) is equivalent to the result of replacing $q_{w \leq S'}$ in the left-hand side with $[w \leq S']$, and $q_{w-w_m \leq S'}$ with $[w - w_m \leq S']$. Since $q_{w \leq S'}$ and $q_{w-w_m \leq S'}$ belong to Q_Ω, this observation implies the derivability of (49) from (48).

The derivability of (49) from (47) will be proved by strong induction on m. If $w \leq 0$ or $w > w_1 + \cdots + w_m$ then, by the definition of $Def(q_{w \leq S})$ and by Lemma 11, (49) is provable in the logic of here-and-there. Assume that $0 < w \leq w_1 + \cdots + w_m$. Then $q_{w \leq S'}$ and $q_{w-w_m \leq S'}$ belong to Q_Ω, and, by the induction hypothesis, the equivalences

$$Def(q_{w \leq S'}) \leftrightarrow [w \leq S']$$

and

$$Def(q_{w-w_m \leq S'}) \leftrightarrow [w - w_m \leq S']$$

are derivable from (47). Consequently, the equivalences

$$q_{w \leq S'} \leftrightarrow [w \leq S']$$

and

$$q_{w-w_m \leq S'} \leftrightarrow [w - w_m \leq S']$$

are derivable from (47) as well. By Lemma 11, this implies the derivability of (49). □

Theorem 2
For any program Ω with weight constraints, $Z \mapsto Z \setminus Q_\Omega$ is a 1–1 correspondence between the answer sets for $[\Omega]^{nn}$ and the answer sets for Ω.

Proof
From Lemmas 10 and 12 we see that $[\Omega]^{nn}$ has the same answer sets as the union of Π and (48). Furthermore, this union is strongly equivalent to the union of $[\Omega]^{nd}$ and (48). Indeed, Π consists of the rules

$$\begin{array}{l} l \leftarrow not\ q_{not\ l}, [C_1]^{nn}, \ldots, [C_n]^{nn}, \\ \bot \leftarrow not\ q_{L_0 \leq S_0}, [C_1]^{nn}, \ldots, [C_n]^{nn}, \\ \bot \leftarrow q_{U_0 < S_0}, [C_1]^{nn}, \ldots, [C_n]^{nn} \end{array}$$

for every rule

$$L_0 \leq S_0 \leq U_0 \leftarrow C_1, \ldots, C_n$$

in Ω and every positive head element l of that rule; $[\Omega]^{nd}$ consists of the rules

$$\begin{array}{l} l \leftarrow not\ not\ l, [C_1], \ldots, [C_n], \\ \bot \leftarrow not\ [L_0 \leq S_0, S_0 \leq U_0], [C_1], \ldots, [C_n]. \end{array}$$

It is easy to derive each of these two programs from the other program and (48) in the logic of here-and-there. Consequently, $[\Omega]^{nn}$ has the same answer sets as the union of $[\Omega]^{nd}$ and (48). By the completion lemma, it follows that $[\Omega]^{nn}$ has the same answer sets as the union of $[\Omega]^{nd}$ and the program

$$\begin{array}{l} \{q_{not\ l} \leftarrow not\ l \ :\ q_{not\ l} \in Q_\Omega\} \cup \\ \{q_{w \leq S} \leftarrow [w \leq S] \ :\ q_{w \leq S} \in Q_\Omega\} \cup \\ \{q_{w < S} \leftarrow [w < S] \ :\ q_{w < S} \in Q_\Omega\}. \end{array}$$

The assertion of Theorem 2 follows now by the lemma on explicit definitions. □

7 Conclusion

The results of this paper show that weight constraints in the sense of (Niemelä and Simons 2000) can be viewed as shorthand for nested expressions. Rules with weight constraints can be equivalently written as sets of nondisjunctive rules. These rules can be further made nonnested, without a significant increase in the size of the program, provided that auxiliary atoms are allowed. Moreover, when all weights are integers from a fixed finite set, this translation leads to a program of about the same size as the original program with weight constraints. These facts, along with the extension of the theory of tight programs proposed in (Erdem and Lifschitz 2003), have led to the creation of the system CMODELS. The ideas of this paper can be also used to prove the strong equivalence of programs with weight constraints.

Acknowledgments

We are grateful to Hudson Turner for his careful reading of a draft of this paper and for many useful comments, and to the anonymous referees for their suggestions. This work was partially supported by National Science Foundation under grant IIS-9732744 and by the Texas Higher Education Coordinating Board under Grant 003658-0322-2001.

References

Yuliya Babovich, Esra Erdem, and Vladimir Lifschitz. Fages' theorem and answer set programming.[9] In *Proc. Eighth Int'l Workshop on Non-Monotonic Reasoning*, 2000.

Wolfgang Bibel and Elmar Eder. A survey of logical calculi. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *The Handbook of Logic in AI and Logic Programming*, volume 1, pages 67–182. Oxford University Press, 1993.

Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

Esra Erdem and Vladimir Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, 2003. To appear.

Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Logic Programming: Proc. Fifth Int'l Conf. and Symp.*, pages 1070–1080, 1988.

Arend Heyting. Die formalen Regeln der intuitionistischen Logik. *Sitz. Berlin*, pages 42–56, 1930.

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.

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

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

[9] `http://arxiv.org/abs/cs.ai/0003042` .

John Lloyd and Rodney Topor. Making Prolog more expressive. *Journal of Logic Programming*, 3:225–240, 1984.

Victor Marek and Jeffrey Remmel. On logic programs with cardinality constraints. In *Proc. NMR-02*, 2002.

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.

David Pearce, Torsten Schaub, Vladimir Sarsakov, Hans Tompits, and Stefan Woltran. A polynomial translation of logic programs with nested expressions into disjunctive logic programs. In *Proc. NMR-02*, 2002.

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.

Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *AI Journal*, 138:181–234, 2002.

Hudson Turner. Strong equivalence made easy: nested expressions and weight constraints. *Theory and Practice of Logic Programming*, 2003. To appear.