Strong Equivalence Made Easy: Nested Expressions and Weight Constraints

HUDSON TURNER
Computer Science Department
University of Minnesota, Duluth
(*e-mail:* hudson@d.umn.edu)

Abstract

Logic programs P and Q are strongly equivalent if, given any program R, programs $P \cup R$ and $Q \cup R$ are equivalent (that is, have the same answer sets). Strong equivalence is convenient for the study of equivalent transformations of logic programs: one can prove that a local change is correct without considering the whole program. Lifschitz, Pearce and Valverde showed that Heyting's logic of here-and-there can be used to characterize strong equivalence for logic programs with nested expressions (which subsume the better-known extended disjunctive programs). This note considers a simpler, more direct characterization of strong equivalence for such programs, and shows that it can also be applied without modification to the weight constraint programs of Niemelä and Simons. Thus, this characterization of strong equivalence is convenient for the study of equivalent transformations of logic programs written in the input languages of answer set programming systems `dlv` and SMODELS. The note concludes with a brief discussion of results that can be used to automate reasoning about strong equivalence, including a novel encoding that reduces the problem of deciding the strong equivalence of a pair of weight constraint programs to that of deciding the inconsistency of a weight constraint program.

KEYWORDS: answer sets, strong equivalence, nested expressions, weight constraints

1 Introduction

Logic programs P and Q are "strongly equivalent" if, given any program R, $P \cup R$ and $Q \cup R$ are equivalent (that is, have the same answer sets). Strong equivalence is important because it allows one to justify changes to one part of a program without considering the whole program. Moreover, as we show, determining that programs are strongly equivalent is no harder (and for some classes of programs may be easier) than determining that they are equivalent.

In a groundbreaking paper, Lifschitz, Pearce and Valverde (2001) used Heyting's logic of here-and-there to characterize strong equivalence of logic programs with nested expressions (Lifschitz et al. 1999). Such "nested programs" subsume the class of extended disjunctive programs (Gelfond and Lifschitz 1991), which can be given as input to the answer set programming system `dlv`.[1]

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

The current note characterizes strong equivalence of nested programs in terms of concepts used in the definition of answer sets. Hence, no knowledge of the logic of here-and-there is required. In (Turner 2001), we showed that this characterization of strong equivalence is easily extended to default logic (Reiter 1980). In the current note, we show that it applies not only to nested programs but also to the weight constraint programs of Niemelä and Simons (2000). Thus it is also convenient for the study of equivalent transformations of logic programs written in the input language of the answer set programming system SMODELS.[2] We also show how to encode the question of strong equivalence of two weight constraint programs *in* a weight constraint program, so that SMODELS can be used to decide strong equivalence.

This note continues as follows. Section 2 defines nested programs. Section 3 establishes the characterization of strong equivalence for nested programs, and Section 4 discusses strongly equivalent transformations of them. Section 5 defines weight constraint programs. Section 6 establishes the characterization of strong equivalence for weight constraint programs, and Section 7 discusses strongly equivalent transformations of *them*. Section 8 compares the approach to strong equivalence presented in this note with that of Lifschitz, Pearce and Valverde. Section 9 discusses results supporting the possibility of automated reasoning about strong equivalence, both for nested programs (via encoding in classical propositional logic) and for weight constraint programs (via encoding in weight constraint programming). [3]

2 Nested Logic Programming

This paper employs the definition of nested programs introduced in (Lifschitz et al. 1999), although the presentation differs in some details.

2.1 Syntax

The words *atom* and *literal* are understood here as in propositional logic. *Elementary formulas* are literals and the 0-place connectives $\bot$ ("false") and $\top$ ("true"). *Formulas* are built from elementary formulas using the unary connective *not* and the binary connectives , (conjunction) and ; (disjunction). A *rule* is an expression of the form

$$F \leftarrow G$$

where F and G are formulas, called the *head* and the *body* of the rule.

A *nested program* is a set of rules.

When convenient, a rule $F \leftarrow \top$ is identified with the formula F.

A program is *nondisjunctive* if the head of each rule is an elementary formula possibly preceded by *not*.

[2] Available at `http://www.tcs.hut.fi/Software/smodels/` .
[3] Much of the material in Sections 3, 4 and 8 is adapted from (Turner 2001).

2.2 Semantics

Let X be a consistent set of literals.

We first define recursively when X *satisfies* a formula F (symbolically, $X \models F$), as follows.

- For elementary F, $X \models F$ iff $F \in X$ or $F = \top$.
- $X \models (F, G)$ iff $X \models F$ and $X \models G$.
- $X \models (F; G)$ iff $X \models F$ or $X \models G$.
- $X \models \mathit{not}\, F$ iff $X \not\models F$.

To continue, X *satisfies* a rule $F \leftarrow G$ if $X \models G$ implies $X \models F$, and X *satisfies* a nested program P if it satisfies every rule in P.

The *reduct* of a formula F relative to X (written F^X) is obtained by replacing every maximal occurrence in F of a formula of the form $\mathit{not}\ G$ with $\bot$ if $X \models G$ and with $\top$ otherwise.[4] The *reduct* of a nested program P relative to X (written P^X) is obtained by replacing the head and body of each rule in P by their reducts relative to X.

Finally, X is an *answer set* for a nested program P if it is minimal among the consistent sets of literals that satisfy P^X.

As discussed in (Lifschitz et al. 1999), this definition agrees with previous versions of the answer set semantics on consistent answer sets (but does not allow for an inconsistent one).

3 Strong Equivalence of Nested Programs

Nested programs P and Q are *equivalent* if they have the same answer sets. They are *strongly equivalent* if, for any nested program R, $P \cup R$ and $Q \cup R$ are equivalent.

Notice that, as an immediate consequence, if P and Q are strongly equivalent, then so are $P \cup R$ and $Q \cup R$.

Definition of SE-model

For nested program P, and consistent sets X, Y of literals with $X \subseteq Y$, the pair (X, Y) is an *SE-model* of P if $Y \models P$ and $X \models P^Y$.[5]

Theorem 1

(Turner 2001) Nested programs are strongly equivalent iff they have the same SE-models.

A proof of Theorem 1 is included here. (Precisely this proof also establishes the similar result for weight constraint programs stated in Section 5.)

We begin with two lemmas.

First notice that a consistent set Y of literals is an answer set for a program P iff (Y, Y) is the unique SE-model of P whose second component is Y. This observation yields the following lemma.

[4] A maximal occurrence in F of a formula of the form $\mathit{not}\ G$ is: a subformula $\mathit{not}\ G$ of F such that there is no subformula $\mathit{not}\ H$ of F which has $\mathit{not}\ G$ as a proper subformula.

[5] This definition is stated in a slightly different form in (Turner 2001). One easily verifies that the two versions are equivalent. The key fact: $Y \models P$ iff $Y \models P^Y$.

Lemma 1
Programs with the same SE-models are equivalent.

Next notice that one can decide whether a pair (X, Y) is an SE-model of a program P by checking whether, for each rule $F \leftarrow G$ in P, Y satisfies $F \leftarrow G$ and X satisfies $F^Y \leftarrow G^Y$. Hence the following.

Lemma 2
The SE-models of a program $P \cup R$ are exactly the SE-models common to programs P and R.

The right-to-left part of the proof of Theorem 1 is easy given these lemmas. The other direction is a bit harder, but the proof of the corresponding result in (Lifschitz et al. 2001) suggests a straightforward construction which also has the virtue of demonstrating that if nested programs P and Q are not strongly equivalent then they can be distinguished by adding rules in which the head is a literal and the body is either a literal or $\top$.

Proof of Theorem 1
Right to left: Assume that programs P and Q have the same SE-models. Take any program R. We need to show that $P \cup R$ and $Q \cup R$ are equivalent. From Lemma 2 we can conclude that $P \cup R$ and $Q \cup R$ have the same SE-models, and so, by Lemma 1, they are equivalent.

Left to right: Assume (without loss of generality) that (X, Y) is an SE-model of program P but not of program Q. We need to show that P and Q are not strongly equivalent. Consider two cases.

Case 1: $Y \not\models Q$. Then $Y \not\models Q \cup Y$, and so Y is not an answer set for $Q \cup Y$. On the other hand, since $Y \models P$ by assumption, it is clear that $Y \models P \cup Y$. It follows that $Y \models (P \cup Y)^Y$. Moreover, no proper subset of Y satisfies $(P \cup Y)^Y = P^Y \cup Y$, which shows that Y is an answer set for $P \cup Y$. Hence P and Q are not strongly equivalent.

Case 2: $Y \models Q$. Take $R = X \cup \{L \leftarrow L' : L, L' \in Y \setminus X\}$. Clearly $Y \models Q \cup R$, and it follows that $Y \models (Q \cup R)^Y$. Let Z be a subset of Y such that $Z \models (Q \cup R)^Y$ $(= Q^Y \cup R)$. By choice of R we know that $X \subseteq Z$, and by assumption $X \not\models Q^Y$, so $X \neq Z$. Hence there is some $L \in Y \setminus X$ that belongs to Z. It follows by choice of R that $Y \setminus X \subseteq Z$. Consequently $Z = Y$, and so Y is an answer set for $Q \cup R$. On the other hand, X is a proper subset of Y that satisfies $P^Y \cup R = (P \cup R)^Y$. So Y is not an answer set for $P \cup R$, and we conclude again that P and Q are not strongly equivalent. $\square$

The form of the respective definitions may seem to suggest that deciding equivalence of nested programs will be easier than deciding strong equivalence. In fact, the opposite is (under the usual assumptions) true. (Similar, independently-obtained complexity results appear in (Pearce et al. 2001; Lin 2002).)

Theorem 2
The problem of determining that two nested programs are equivalent is $\mathbf{\Pi_2^P}$-hard. The problem of determining that they are strongly equivalent belongs to **coNP**.

Proof

For the first part, we show that the complementary problem is $\mathbf{\Sigma_2^P}$-hard. Eiter and Gottlob (1993) showed that it is $\mathbf{\Sigma_2^P}$-hard to determine that a "disjunctive" logic program has an answer set. This result extends to nested programs, which include the disjunctive programs as a special case. To show that a nested program is not equivalent to the program $\{\bot\}$, one must show that it has an answer set. So determining that two nested programs are not equivalent is $\mathbf{\Sigma_2^P}$-hard.

For the second part, we observe that, given Theorem 1, the complementary problem belongs to **NP**. That is, given two nested programs, guess a pair (X, Y) of consistent sets of literals, and verify in polynomial time that (X, Y) is an SE-model of exactly one of the two programs. $\square$

For $\neg$-free programs (that is, programs in which $\neg$ does not occur), the characterization of strong equivalence can be simplified: one can restrict attention to the *positive* SE-models—those in which only atoms appear.

For any set X of literals, let X^+ be the set of atoms that belong to X.

Lemma 3

Let P be a $\neg$-free program. For any consistent sets X, Y of literals such that $X \subseteq Y$, (X, Y) is an SE-model of P iff (X^+, Y^+) is.

The proof of Lemma 3, which is straightforward, is omitted. The following is an easy consequence of Lemma 3 and Theorem 1.

Theorem 3

A pair of $\neg$-free nested programs are strongly equivalent iff they have the same positive SE-models.

4 Equivalent Transformations of Nested Programs

To demonstrate the use of Theorem 1, let us first consider an example discussed at length in (Lifschitz et al. 2001). For any formulas F and G, programs P_1 and P_2 below have the same SE-models.

$$\begin{array}{ll} F; G & F \leftarrow not\, G \\ \bot \leftarrow F, G & G \leftarrow not\, F \\ & \bot \leftarrow F, G \end{array}$$

To see this, take any pair (X, Y) of consistent sets of literals such that $X \subseteq Y$, and consider four cases.

Case 1: $Y \models (F, G)$. Then $Y \not\models P_1$ and $Y \not\models P_2$, so (X, Y) is not an SE-model of P_1 or P_2.
Case 2: $Y \models (F, not\, G)$. Then $Y \models P_1$ and $Y \models P_2$. Since $Y \models not\, G$, $Y \not\models G$, and so $Y \not\models G^Y$. Since *not* does not occur in G^Y and $X \subseteq Y$, $X \not\models G^Y$. We can conclude that $X \models P_1^Y$ iff $X \models F^Y$ iff $X \models P_2^Y$. So (X, Y) is an SE-model of P_1 iff it is an SE-model of P_2.
Case 3: $Y \models (not\, F, G)$. Symmetric to previous case.
Case 4: $Y \models (not\, F, not\, G)$. Similar to first case.

It follows by Theorem 1 that in any program that contains P_1, P_1 can be safely replaced by P_2, thus eliminating an occurrence of disjunction in the heads of rules. (This result generalizes a theorem from (Erdem and Lifschitz 1999).)

On the other hand, Theorem 1 can also be used to show that no nondisjunctive program is strongly equivalent to the program $\{p; q\}$. (This was suggested as a challenge problem by Vladimir Lifschitz.) We begin with an easily verified observation. Let P be a nondisjunctive program with no occurrences of *not*. The set of consistent sets of literals satisfying P is closed under intersection.

Proposition 1
No nondisjunctive program is strongly equivalent to $\{p; q\}$.

Proof
Let P be a program strongly equivalent to $\{p; q\}$. Notice that both $\{p\}$ and $\{q\}$ satisfy $\{p; q\}^{\{p,q\}}$, but $\emptyset$ doesn't. By Theorem 1, the same is true of $P^{\{p,q\}}$. It follows by the preceding observation that $P^{\{p,q\}}$ is not nondisjunctive, and consequently neither is P. $\square$

Next we state a replacement theorem for nested programs. For this we need the following definitions. Formulas F and G are *equivalent* relative to program P if, for every SE-model (X, Y) of P, $X \models F^Y$ iff $X \models G^Y$. An occurrence of a formula is *regular* unless it is an atom preceded by $\neg$.

Theorem 4
(Turner 2001) Let P be a nested program, and let F and G be formulas equivalent relative to P. For any nested program Q, and any nested program Q' obtained from Q by replacing regular occurrences of F by G, programs $P \cup Q$ and $P \cup Q'$ are strongly equivalent.

The restriction to regular occurrences is essential. For example, formulas p and q are equivalent relative to program $P_3 = \{p \leftarrow q,\ q \leftarrow p\}$, yet programs $P_3 \cup \{\neg p\}$ and $P_3 \cup \{\neg q\}$ are not strongly equivalent.

Theorem 4 is a more widely-applicable version of Proposition 3 from (Lifschitz et al. 1999). There we defined equivalence of formulas more strictly, and did not make it relative to a program. We also used a notion of "equivalence" of programs stronger than strong equivalence. Many formula equivalences are proved there (see Proposition 4, (Lifschitz et al. 1999)), and of course they also hold under this new (weaker) definition (relative to the empty program). Thus, Theorem 4 implies, for instance, that replacing subformulas of the form *not* (F, G) with *not* F; *not* G yields a strongly equivalent program.

For another example using Theorem 4, observe that for any program Q, and any program Q' obtained from Q by replacing occurrences of *not* F by G and/or *not* G by F, programs $P_2 \cup Q$ and $P_2 \cup Q'$ are strongly equivalent.

5 Weight Constraint Programming

This presentation is adapted from (Ferraris and Lifschitz 2001), and extends slightly the definition of weight constraint programming from (Niemelä and Simons 2000).

5.1 Syntax

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

$$e = w \tag{1}$$

where e is a rule element and w is a nonnegative real number (a "weight"). The part "$= w$" of (1) can be omitted if w is 1. A *weight constraint* is an expression of the form

$$L \leq S \leq U \tag{2}$$

where S is a finite set of weight assignments, and each of L, U is a real number or one of the symbols $-\infty$, $+\infty$. The part "$L \leq$" can be omitted from (2) if L is $-\infty$; similarly, the part "$\leq U$" can be omitted if U is $+\infty$. A *WCP rule* 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 weight constraints such that C_0 does not contain negative rule elements. We call C_0 the *head* of (3), and the rule elements that occur in the head are called the *head literals* of (3).

A *weight constraint program* is a set of WCP rules.

This syntax becomes a generalization of the syntax of "extended" logic programs, introduced in (Gelfond and Lifschitz 1990), if we allow a rule element e to stand for the weight constraint $1 \leq \{e\}$.

In weight constraint programs, let $\bot$ stand for the weight constraint $1 \leq \{\}$. When convenient, a WCP rule $C \leftarrow$ is identified with the weight constraint C.

5.2 Semantics

Let X be a consistent set of literals.

For any finite set S of weight assignments, let

$$v(S, X) = \sum_{\substack{e=w \,\in\, S \\ X \models e}} w \cdot$$

We say X *satisfies* a weight constraint $L \leq S \leq U$ if $L \leq v(S,X) \leq U$. To continue, X satisfies a WCP rule $C_0 \leftarrow C_1, \ldots, C_n$ if X satisfies C_0 whenever X satisfies all of $C_1, \ldots, C_n$, and X satisfies a weight constraint program P if it satisfies every rule in P.

For any weight constraint that can be written in the form $L \leq S$, its *reduct* $(L \leq S)^X$ with respect to X is the weight constraint $L^X \leq S'$ where

- $S' = \{e = w \in S \mid e \text{ is a positive rule element}\}$, and
- $L^X = L - v(S \setminus S', X)$.

The *reduct* of a WCP rule

$$L_0 \leq S_0 \leq U_0 \leftarrow L_1 \leq S_1 \leq U_1, \ldots, L_n \leq S_n \leq U_n \tag{4}$$

with respect to X is the weight constraint program consisting of all rules

$$e \leftarrow (L_1 \leq S_1)^X, \ldots, (L_n \leq S_n)^X$$

such that

- e is a head literal of (4),
- $X \models e$, and
- $X \models S_i \leq U_i$ for all $i \in \{1, \ldots, n\}$.

The *reduct* P^X of a weight constraint program P with respect to X is the union of the reducts with respect to X of all rules in P.

Finally, X is an answer set for a weight constraint program P if X satisfies P and no proper subset of X satisfies P^X.[6]

The semantics of nested programs and weight constraint programs agree wherever their syntax overlaps.

6 Strong Equivalence for Weight Constraint Programs

The definitions of equivalence, strong equivalence and SE-models for weight constraint programs are as they were for nested programs.

Theorem 5
Weight constraint programs are strongly equivalent iff they have the same SE-models.

As mentioned previously, the proof of Theorem 1 applies, without change, to this theorem also.[7]

In addition, as with nested programs, the proof of Theorem 5 shows that weight constraint programs that are not strongly equivalent can be distinguished by adding rules that either can be represented by a literal or can be written $L \leftarrow L'$, where L and L' are literals.

Moreover, it is clear that the **coNP** complexity of deciding strong equivalence carries over to weight constraint programs. (The same easy argument applies.)

And also as with nested programs, when programs are $\neg$-free (that is, have no occurrences of $\neg$), we can restrict attention to positive SE-models (in which only atoms appear). Lemma 3, stated previously in the context of nested programs, holds also for weight constraint programs, and together with Theorem 5 yields:

Theorem 6
A pair of $\neg$-free weight constraint programs are strongly equivalent iff they have the same positive SE-models.

[6] The requirement that $X \models P$, which does not appear explicitly in the definition of answer sets for nested programs, is necessary here because, for weight constraint programs, $X \models P^X$ does not imply that $X \models P$. Note that the converse *does* still hold.

[7] This easy correspondence implies that there is no difficulty in allowing a more general class of programs that can be formed as the union of a nested program and a weight constraint program. No doubt more elaborate hybrids are possible as well, some of them quite straightforward.

7 Equivalent Transformations of Weight Constraint Programs

We start with an adaptation of the first example from Section 4. For any literals L and L', the two programs shown below have the same SE-models.

$$\begin{array}{rl} L \leftarrow \mathit{not}\, L' & \qquad 1 \leq \{L, L'\} \leq 1 \\ L' \leftarrow \mathit{not}\, L & \\ \bot \leftarrow L, L' & \end{array}$$

This is easily verified, much as was done for the similar nested program example.

Ferraris and Lifschitz (2001) introduced a translation from weight constraint programs to nested programs, and argued that this translation is interesting in part because it provides, indirectly, a method for reasoning about strong equivalence of weight constraint programs. Next we consider the main example from that paper.

We are interested in the n-Queens program consisting of the following rules, where $i, i', j, j' \in \{1, \ldots, n\}$.

$$1 \leq \{q(1,j), \ldots, q(n,j)\} \leq 1 \tag{5}$$

$$1 \leq \{q(i,1), \ldots, q(i,n)\} \leq 1 \tag{6}$$

$$\bot \leftarrow q(i,j), q(i',j') \qquad (|i-i'| = |j-j'|) \tag{7}$$

Intuitively, (5) expresses that, for each column j, there is a queen in exactly one row. Similarly, (6) expresses that, for each row i, there is a queen in exactly one column. Finally, (7) stipulates that no two queens occupy a common diagonal.

We wish to verify that (6) can be equivalently replaced by the following.

$$\bot \leftarrow q(i,j), q(i,j') \qquad (j < j') \tag{8}$$

What we'll show is that program P consisting of rules (5) and (6) is strongly equivalent to program Q consisting of rules (5) and (8).

Observe first that the positive SE-models of the rules (5) are exactly the pairs (X, X) where

$$X = \{q(i_1, 1), \ldots, q(i_n, n)\}$$

with $i_1, \ldots, i_n \in \{1, \ldots, n\}$. To see this, notice that

- the rules (5) are satisfied by all and only such sets X, and
- the reduct of these rules with respect to such an X is not satisfied by any proper subset of X (and in fact can be written as X).

Next observe that such an X satisfies the rules (6) iff all of $i_1, \ldots, i_n$ are different. Finally, such an X satisfies the rules (8) under exactly the same conditions. We can conclude that programs P and Q have the same positive SE-models, and by Theorem 6 they are strongly equivalent.

8 SE-Models and the Logic of Here-and-There

Lifschitz, Pearce and Valverde (2001) identify $\neg$-free nested logic program rules with formulas in Heyting's logic of here-and-there, and show that $\neg$-free nested programs are strongly equivalent iff they are equivalent in the logic of here-and-there. They also explain that this result can be extended to all nested programs (including those in which $\neg$ occurs) in the standard fashion, by translating a program with occurrences of $\neg$ into one without. (See their paper for details.)

According to their definitions, an *HT-interpretation* is a pair (I^H, I^T) of sets of atoms, with $I^H \subseteq I^T$. Without going into details, we can observe that they define when an HT-interpretation is a model of a $\neg$-free nested program in the sense of the logic of here-and-there. Although it is not done here, one can verify that their Lemmas 1 and 2 together imply the following.

Proposition 2
For any $\neg$-free nested program P, (X, Y) is a positive SE-model of P iff (X, Y) is a model of P in the logic of here-and-there.

Not surprisingly, it then follows from Theorem 3 that these two characterizations of strong equivalence are essentially equivalent with regard to $\neg$-free nested logic programs. A key advantage though of the SE-models approach is that it is easily extended to other, similar nonmonotonic formalisms. In Section 6, the SE-models characterization of strong equivalence was extended to weight constraint programs (without altering the definition of SE-model or the proof of the strong equivalence theorem). In (Turner 2001), a similar notion of SE-model was used to characterize strong equivalence for default logic. The extension in this case was also easy.[8]

Even when we consider strong equivalence only for nested programs, it seems that the SE-models and the here-and-there characterizations have different strengths.

One advantage of the SE-models approach is its relative simplicity. The definition is quite straightforward, based on concepts already introduced in the definition of answer sets. This in turn simplifies the proof of the strong equivalence theorem.

The definition of an SE-model for nested programs takes advantage of the special status of the symbol $\leftarrow$ in usual definitions of logic programming. By comparison, the logic of here-and-there treats $\leftarrow$ as just another connective, and even defines *not* in terms of it—*not* F is understood as an abbreviation for $\bot \leftarrow F$. The possibility of nested occurrences of $\leftarrow$ complicates the truth definition considerably.

It is important to note, though, that this complication takes a familiar form—the truth definition in the logic of here-and-there uses standard Kripke models. In fact, they are a special case of Kripke models for intuitionistic logic (which is, accordingly, slightly weaker). Thus, such an approach brings with it a range of associations that may help clarify intuitions about the meaning of connectives $\leftarrow$ and *not* in logic programming.

Even if we consider only convenience in the study of strong equivalence (or similar properties), the logic of here-and-there offers a potential advantage: it is a logic with known identities, deduction rules, and such, which can be used to reason about strong equivalence in particular cases.

[8] In fact, although we do not go into details here, the SE-models characterization of strong equivalence is also easily adapted to the causal theories formalism of (McCain and Turner 1997) and the modal causal logic UCL of (Turner 1999). Similar characterizations are likely for other nonmononotic formalisms based on such fixpoint semantics.

Nonetheless, when we wish to apply strong equivalence results, it seems likely that a model-theoretic argument based on SE-models will often be easier than a proof-theoretic argument using known properties of the logic of here-and-there.

9 Toward automated reasoning about strong equivalence

It may be desirable to use automated methods to reason about strong equivalence. One possibility would be to employ general-purpose tools for the logic of here-and-there, but we may instead wish to take advantage of recent results regarding encodings of strong equivalence of nested programs in classical propositional logic (Pearce et al. 2001; Lin 2002). The existence of such encodings is suggested by the **coNP** complexity of the decision problem, and indeed they are not hard to find.[9] An encoding like those in (Pearce et al. 2001; Lin 2002) is specified below. After this, we specify a similar, new encoding of strong equivalence of weight constraint programs, this time *in* a weight constraint program.

9.1 Strong equivalence of nested programs as unsatisfiability

The key is a translation that maps a nested program to a propositional theory whose models are in one-to-one correspondence with the SE-models of the program.[10] Here it is convenient to restrict consideration to $\neg$-free programs.

Consider any $\neg$-free nested program P. The first step is to augment the signature—for each atom A of the language of P add a new atom A'. A classical interpretation I of the augmented language corresponds to a pair (I_{new}, I_{old}), where I_{old} consists of the original atoms that are true in I and I_{new} consists of the original atoms A such that $I \models A'$. Then for any rule $F \leftarrow G$, let $pl(F \leftarrow G)$ stand for $pl(G) \supset pl(F)$, where $pl(F)$ and $pl(G)$ are obtained from F and G by replacing occurrences of *not* with $\neg$, ; with $\vee$ and , with $\wedge$, yielding a formula of classical logic. For any classical propositional formula ϕ, let ϕ' be obtained from ϕ by replacing each occurrence of an atom A of the original language that is not in the scope of $\neg$ with its counterpart A'. Let $pl(P)$ be the following classical propositional theory.

$$\{pl(r) : r \in P\} \cup \{pl(r)' : r \in P\} \cup \{A' \supset A : A \text{ is an atom of the original language}\}$$

It is straightforward to verify that $I \models pl(P)$ iff (I_{new}, I_{old}) is an SE-model of P. Moreover, every positive SE-model of P can be written in the form (I_{new}, I_{old}) for some interpretation I of the language of $pl(P)$.

Given this encoding of the positive SE-models of a $\neg$-free nested program, we can construct (via Theorem 3), for any finite $\neg$-free nested programs P and Q,

[9] More surprising is a related result due to Lin (2002) showing that strong equivalence of disjunctive logic programs with variables and constants (but without proper functions) is also **coNP**, despite the fact that equivalence for such programs is undecidable!

[10] The easy proof in Lin's paper (for disjunctive programs only) is based directly on this idea. Pearce et al. (2001) show that the same kind of translation can be applied to any theory in the logic of here-and-there, yielding a classical propositional theory whose models correspond to the models of the original theory (in the logic of here-and-there).

a classical propositional formula that is satisfiable iff P and Q are not strongly equivalent. Abusing notation, take $pl(P)$ to stand for the (finite) conjunction of its elements, and do the same for $pl(Q)$. Then P and Q are strongly equivalent iff the formula $pl(P) \not\equiv pl(Q)$ is unsatisfiable.

9.2 Strong equivalence of weight constraint programs as inconsistency (in weight constraint programming)

One could devise a similar encoding in classical propositional logic for the SE-models of a weight constraint program (and then decide strong equivalence of a pair of finite weight constraint programs by deciding unsatisfiability of a classical propositional formula, as above). Unfortunately this would require a translation of weight constraints into classical propositional logic, which would in general be rather costly. In light of this, it may be preferable instead to use an encoding *in* a weight constraint program. The crucial step—capturing the SE-models of a weight constraint program as answer sets of a weight constraint program—is quite easy. But the subsequent step—encoding the equivalence of two weight constraint programs as inconsistency of a weight constraint program—requires some additional work, as we will see. We again restrict consideration to $\neg$-free programs.

Consider any $\neg$-free weight constraint program P. Augment the language as before, using similar notation (X_{new}, X_{old}) for the pair of sets of atoms in the original language corresponding to a set X of atoms in the augmented language. (So $X = X_{old} \cup \{A' : A \in X_{new}\}$.) Let P' be obtained from P by replacing each occurrence of an atom A not preceded by *not* with A'. Let $wc(P)$ be the program obtained by adding to $P \cup P'$ the rules

$$\bot \leftarrow A', not\ A\,,$$
$$\{A\}\,,$$
$$\{A'\}\,,$$

for each atom A of the original language. Notice that X is an answer set for $wc(P)$ iff $X \models wc(P)$, since inclusion of the rules of forms $\{A\}$ and $\{A'\}$ effectively renders every atom in the language of $wc(P)$ abducible. Notice also that, as before, it is straightforward to verify that $X \models wc(P)$ iff (X_{new}, X_{old}) is an SE-model of P, and that, moreover, every positive SE-model of P can be written (X_{new}, X_{old}) for some subset X of the atoms of the language of $wc(P)$.

Given this encoding of the positive SE-models of a $\neg$-free weight constraint program, Theorem 6 allows us to reduce the problem of deciding the strong equivalence of $\neg$-free weight constraint programs P and Q to the problem of deciding the equivalence of weight constraint programs $wc(P)$ and $wc(Q)$. We conclude by showing that this latter question can, in turn, be encoded in weight constraint programming.

Recently, Janhunen and Oikarinen (2002) investigated such encodings, but their results do not cover all programs we are interested in. On the other hand, our programs $wc(P)$, $wc(Q)$ are unusual: their answer sets are simply the sets of atoms that satisfy them, because they make all atoms abducible (so to speak). So for our purposes it will be sufficient to describe an encoding (in weight constraint programming) of the following question: Are two $\neg$-free weight constraint programs (in the same language) satisfied by exactly the same sets of atoms?

To this end, we will first define a transformation that takes any $\neg$-free weight constraint program P to a program $not(P)$ whose answer sets, roughly speaking, correspond to the sets of atoms that do not satisfy P. Let $\mathbf{A}$ denote the set of all atoms in the language of P. The language of $not(P)$ is obtained by adding to $\mathbf{A}$ a new atom *witness*, as well as a new atom $h(C_0)$ for each weight constraint C_0 that appears at least once as the head of a rule in P. For each rule $C_0 \leftarrow C_1, \ldots, C_n$ of P, program $not(P)$ includes the rules

$$h(C_0) \leftarrow C_0\,, \tag{9}$$

$$witness \leftarrow not\, h(C_0), C_1, \ldots, C_n\,\cdot \tag{10}$$

Program $not(P)$ also includes, for every atom $A \in \mathbf{A}$, the rule

$$\{A\} \tag{11}$$

and finally the rule

$$\bot \leftarrow not\ witness\,\cdot \tag{12}$$

Notice that the rules (11) make the atoms in $\mathbf{A}$ abducible, while the rules (9) and (10) can provide support only for atoms not in $\mathbf{A}$. In light of these observations it is not difficult to verify that, for every subset X of $\mathbf{A}$, the program consisting just of rules (9), (10) and (11) will have an answer set obtained from X by adding: (i) the atom $h(C_0)$ for each head C_0 from P such that $X \models C_0$, and (ii) the atom *witness* if there is a rule from P that X does not satisfy. Moreover, all answer sets of (9)–(11) can be obtained in this way. The effect of adding rule (12) then is just to eliminate those answer sets for which $X \models P$. Consequently, for every subset X of $\mathbf{A}$, $X \not\models P$ iff there is an answer set Y for $not(P)$ such that $Y \cap \mathbf{A} = X$.

Now consider a second $\neg$-free weight constraint program Q in the same language as P. Again because program $not(P)$ makes all atoms in $\mathbf{A}$ abducible, we can conclude that, for any subset X of $\mathbf{A}$, $X \not\models P$ and $X \models Q$ iff there is an answer set Y for $not(P) \cup Q$ such that $Y \cap \mathbf{A} = X$. It follows that P and Q are satisfied by exactly the same sets of atoms iff both $not(P) \cup Q$ and $P \cup not(Q)$ are inconsistent (that is, have no answer sets).

As a last step in this construction, it is straightforward to combine two weight constraint programs into a single program that is consistent iff at least one of the original two is. Again let's call these programs P and Q. Add new atoms p and q to their common language. Add p to the body of each rule in P; add q to the body of each rule in Q. Take the resulting rules and add to them one more rule: $1 \leq \{p, q\} \leq 1$. Let's call the resulting program $or(P, Q)$. It has an answer set iff at least one of P and Q does.

So, summarizing the result of this subsection, we can decide strong equivalence of arbitrary $\neg$-free weight constraint programs P and Q (in the same language) by deciding the inconsistency of the $\neg$-free weight constraint program

$$or\,(not(se(P)) \cup se(Q), se(P) \cup not(se(Q)))\,\cdot$$

That is, this program has no answer sets iff P and Q are strongly equivalent.

Acknowledgements

Many thanks to Vladimir Lifschitz, who showed me drafts of his related papers and encouraged me to write up what I noticed. Thanks to Alessandro Provetti for a useful conversation about automated reasoning about strong equivalence, and to the anonymous referees for helpful suggestions. This work partially supported by NSF Career Grant #0091773.

References

EITER, T. AND GOTTLOB, G. 1993. Complexity results for disjunctive logic programming and application to nonmonotonic logics. In *Logic Programming: Proceedings of the 1993 International Symposium*. 266–278.

ERDEM, E. AND LIFSCHITZ, V. 1999. Transformations of logic programs related to causality and planning. In *Logic Programming and Non-monotonic Reasoning: Proc. Fifth Int'l Conf. (Lecture Notes in Artificial Intelligence 1730)*. 107–116.

FERRARIS, P. AND LIFSCHITZ, V. 2001. Weight constraints as nested expressions. Available at `www.cs.utexas.edu/users/vl/papers.html`.

GELFOND, M. AND LIFSCHITZ, V. 1990. Logic programs with classical negation. In *Logic Programming: Proc. of the 7th Int'l Conference*, D. Warren and P. Szeredi, Eds. 579–597.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–385.

JANHUNEN, T. AND OIKARINEN, E. 2002. Testing the equivalence of logic programs under stable model semantics. In *Logics in Artificial Intelligence: Proc. 8th European Conference (JELIA'02)*. 493–504.

LIFSCHITZ, V., PEARCE, D., AND VALVERDE, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic 2*, 526–541.

LIFSCHITZ, V., TANG, L., AND TURNER, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25,* 2–3, 369–390.

LIN, F. 2002. Reducing strong equivalence of logic programs to entailment in classical logic. In *Proc. of KR'02*. 170–176.

MCCAIN, N. AND TURNER, H. 1997. Causal theories of action and change. In *Proc. of AAAI-97*. 460–465.

NIEMELÄ, I. AND SIMONS, P. 2000. Extending the SMODELS system with cardinality and weight constraints. In *Logic-Based Artificial Intelligence*, J. Minker, Ed. Kluwer, 491–521.

PEARCE, D., TOMPITS, H., AND WOLTRAN, S. 2001. Encodings for equilibrium logic and logic programs with nested expressions. In *Proc. of the 10th Portuguese Conf. on AI (Lecture Notes in Artificial Intelligence 2258)*. 306–320.

REITER, R. 1980. A logic for default reasoning. *Artificial Intelligence 13,* 1,2, 81–132.

TURNER, H. 1999. A logic of universal causation. *Artificial Intelligence 113*, 87–123.

TURNER, H. 2001. Strong equivalence for logic programs and default theories (made easy). In *Logic Programming and Nonmonotonic Reasoning: Proc. of Sixth Int'l Conf. (Lecture Notes in Artificial Intelligence 2173)*. 81–92.